Computer Modeling of Wideband Quad-Ridge Horn Microwave Antenna Feeds

Gabe A. Cohn

RF and Microwave Group
California Institute of Technology

GabeCohn@caltech.edu

September 21, 2007

Abstract

This paper discusses the modeling and simulation of wideband quad-ridge microwave feeds made by ETS-Lindgren. Computer Modeling is discussed using QuickWave and CST Microwave Studio. In addition, several external postprocessing functions are discussed. Finally, the results of the computer simulations are presented. This document serves as a complete guide to modeling using the discussed software packages. In addition, all code which I wrote to simplify the modeling and postprocessing tasks are included and fully documented.

Table of Contents

1 Background on Lindgren Quad-Ridge Horn Feeds 4
1.1 ETS-Lindgren 3164-05 (2-18 GHz) 5
1.1.1 Data Sheet 5
1.1.2 Return Loss 8
1.1.3 Patterns for Feed Only 9
1.1.4 Patterns for Feed in Radiation Shield 10
1.1.5 Patterns for Feed in Absorber Lined Radiation Shield 12
1.2 ETS-Lindgren 3164-06 (0.3-6 GHz) 14
1.2.1 Data Sheet 15
1.2.2 Return Loss 16
1.2.3 Patterns from Data Sheet 17

2 Modeling Software 18
2.1 QuickWave 19
2.1.1 Features 19
2.1.2 Basic Usage Information 19
2.1.3 Problems, Difficulties, and Errors 22
2.2 CST Microwave Studio 23
2.2.1 Features 23
2.2.2 Basic Usage Information 23
2.2.3 Problems, Difficulties, and Errors 26
2.3 Comparison 26
2.3.1 QuickWave Advantages 27
2.3.2 CST Advantages 27
2.3.3 Conclusions 27

3 CAD Models of Lindgren Feeds 28
3.1 Mechanical Drawings and Measurements 29
3.1.1 Measurements from 3164-05 Inventor File 29
3.1.2 3164-05 Physical Measurements 33
3.1.3 3164-06 Physical Measurements 37
3.2 Modeling Microwave Absorber 41
3.3 QuickWave Model 42
3.3.1 How to build the model 42
3.3.2 Scripts to create QuickWave UDO code 43
3.3.3 QuickWave UDO Code 57
3.4 CST Model 96
3.4.1 How to build the model 96

4 Simulation Outputs 102
4.1 S-Parameters 103
4.1.1 Obtaining S-Parameters in QuickWave 103
4.1.2 Obtaining S-Parameters in CST 103
4.2 Far-field Patterns 104
4.2.1 Running Patterns in QuickWave 104
4.2.2 Running Patterns in CST 105
4.2.3 Standard File Format for Patterns 115
4.2.4 Converting QuickWave Output to Standard Format 115
4.2.5 Converting CST Patterns to Standard Format 119
4.2.6 Converting Measured Patterns to Standard Format 123
4.3 E-Fields and H-Fields 123
4.3.1 Viewing and Exporting Fields in QuickWave 123
4.3.2 Viewing and Exporting Fields in CST 123

5 External Postprocessing 125
5.1 Convert Pattern Data to Co- and Cross-Polar Form 126
5.1.1 coxpol.m 126
5.2 Plotting Far-field Patterns 131
5.2.1 plot_pattern.m 131
5.2.2 auto_patterns.m 136
5.2.3 auto_cmp_patterns.m 139
5.3 Phase Center Analysis 142
5.3.1 phase_center.m 142
5.3.2 change_ref.m 146
5.3.3 auto_pc.m 151
5.4 Computation of Radiated Power, Taper and Spillover Efficiencies 155
5.4.1 eff.m 155

6 Results 164
6.1 Comparison between QuickWave and CST 165
6.1.1 Return Loss 165
6.1.2 Far-field Patterns for Feed Only 167
6.1.3 Far-field Patterns for Feed in Radiation Shield 175
6.1.4 Far-field Patterns for Feed in Absorber Lined Rad. Shield 184
6.2 Data from QuickWave Model of Inventor File Design (1-18 GHz) 193
6.2.1 Far-field Patterns for Feed Only 194
6.2.2 Far-field Patterns for Feed in Radiation Shield 196
6.2.3 Far-field Patterns for Feed in Absorber Lined Rad. Shield 198

6.2.4 Phase Center 200
6.2.5 Efficiencies for Feed Only 201
6.2.6 Efficiencies for Feed in Radiation Shield 202
6.2.7 Efficiencies for Feed in Absorber Lined Rad. Shield 203
6.3 Data from CST Model of 3164-05 Design (1-18 GHz) 204
6.3.1 Return Loss 204
6.3.2 Far-field Patterns for Feed Only 206
6.3.3 Far-field Patterns for Feed in Radiation Shield 215
6.3.4 Far-field Patterns for Feed in Absorber Lined Rad. Shield 225
6.3.5 Phase Center 235
6.3.6 Efficiencies for Feed Only 236
6.3.7 Efficiencies for Feed in Radiation Shield 237
6.3.8 Efficiencies for Feed in Absorber Lined Rad. Shield 238
6.4 Data from CST Model of 3164-06 Design (0.3-6 GHz) 239
6.4.1 Return Loss 239
6.4.2 Far-field Patterns for Feed Only 241
6.4.3 Far-field Patterns for Feed in Radiation Shield 242
6.4.4 Far-field Patterns for Feed in Absorber Lined Rad. Shield 244
6.4.5 Phase Center 246
6.4.6 Efficiencies for Feed Only 247
6.4.7 Efficiencies for Feed in Radiation Shield 248
6.4.8 Efficiencies for Feed in Absorber Lined Rad. Shield 249
6.5 Effect of Fin Thickness 250
6.5.1 Far-field Patterns for 3.615 mm Fin 251
6.5.2 Far-field Patterns for 1.372 mm Fin 253
6.5.3 Far-field Patterns for 1.372 mm Fin with 3.615 mm at Slot 255
6.5.4 Far-field Patterns for 3.615 mm Fin with 3.615 mm Edge 257
6.5.5 Comparison of Far-field Patterns 259

7 References 266

1

Background on Lindgren Quad-Ridge Horn Feeds

1.1	ETS-Lindgren 3164-05 (2-18 GHz)	5
1.1.1	Data Sheet	5
1.1.2	Return Loss	8
1.1.3	Patterns for Feed Only	9
1.1.4	Patterns for Feed in Radiation Shield	10
1.1.5	Patterns for Feed in Absorber Lined Radiation Shield	12
1.2	ETS-Lindgren 3164-06 (0.3-6 GHz)	14
1.2.1	Data Sheet	15
1.2.2	Return Loss	16
1.2.3	Patterns from Data Sheet	17

This section gives basic information about the two ETS-Lindgren feeds that are discussed in this document. Data is reported from the manufacturer's data sheet. In addition, measured data is reported from the actual feeds.

1.1 ETS-Lindgren 3164-05 (2-18 GHz)

The ETS-Lindgren 3164-05 quad-ridge horn feed was designed by V. Rodriguez [1-1]. It has an operation frequency of 2-18 GHz. It will be used as the feed for the upper frequency range of the Goldstone Apple Valley Radio Telescope (GAVRT) [1-2]. The entire receiver system is described in [1-3]. The following image shows the 3164-05 feed.

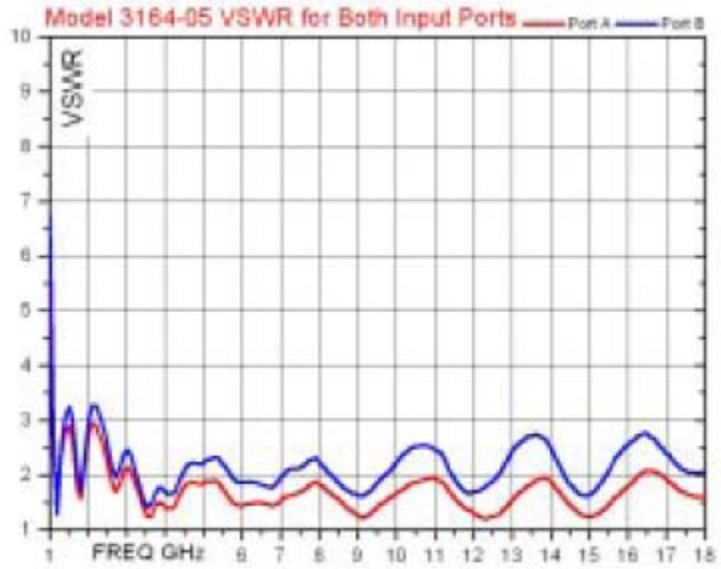
Figure 1.1 a

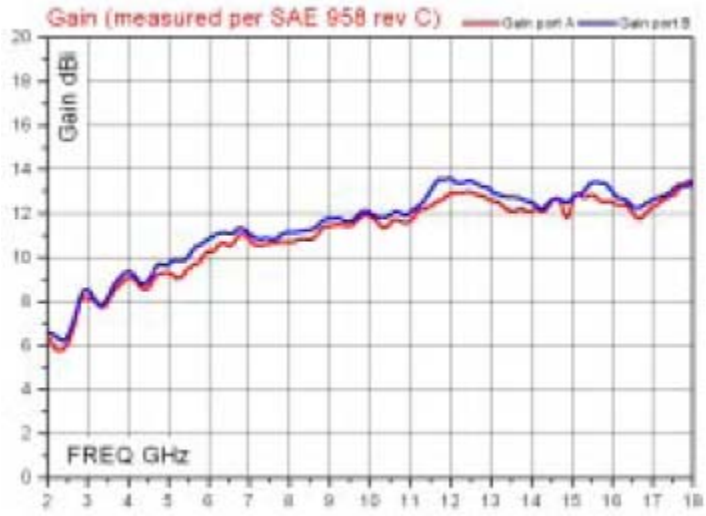
1.1.1 Data Sheet

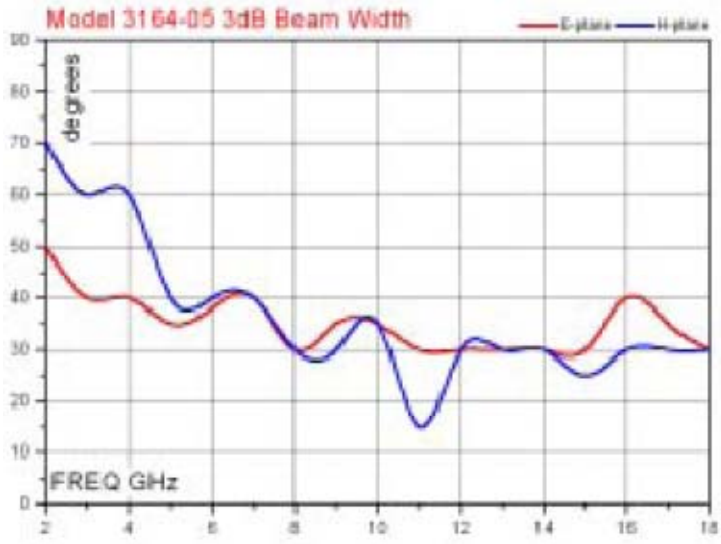
The following data is from the ETS-Lindgren 3164-05 data sheet [1-4]:

> The Model 3164-05 Open Boundary Quadridge Horn is the newest in a series of quadridge horns from ETS-Lindgren. The “open boundary” design with its absence of side plates makes this antenna unique in both appearance and performance. Numerically modeled, the Model 3164’s open boundary design is similar to two Vivaldi PCB antennas placed orthogonally to each other. The antenna’s surprisingly compact size offers improved pattern and gain when

compared with enclosed quadridge horns of similar dimensions. The compact size also means there is only small shift on the Model 3164's phase center as frequency changes. The Model 3164-05 has exceptional bandwidth. While the frequency band for optimum performance is 2 GHz to 18 GHz, the antenna is usable from 1.5 GHz. Two orthogonally placed input feeds allow this antenna to generate both linear and circular polarized measurements across the entire frequency band.



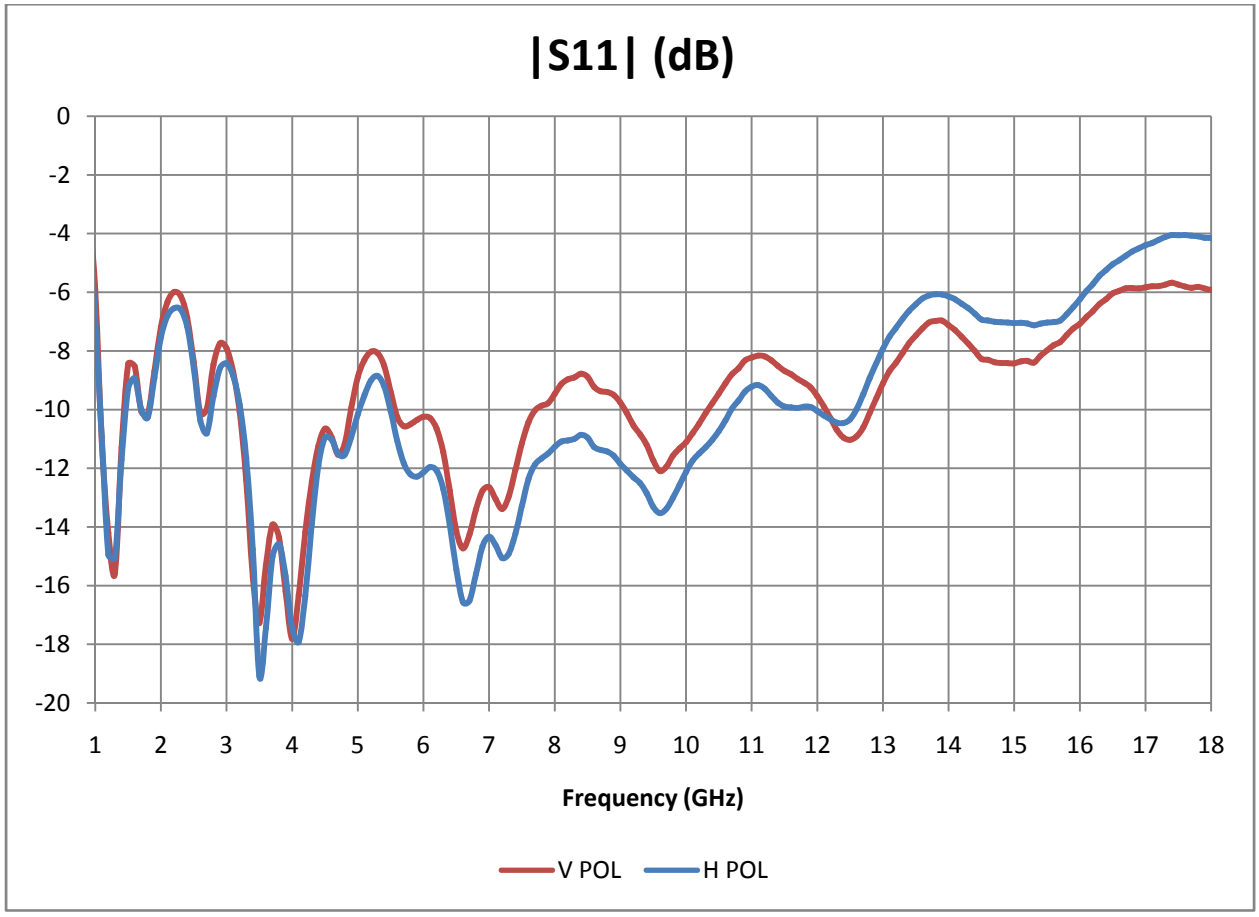




For more information, please see the user's manual [1-5].

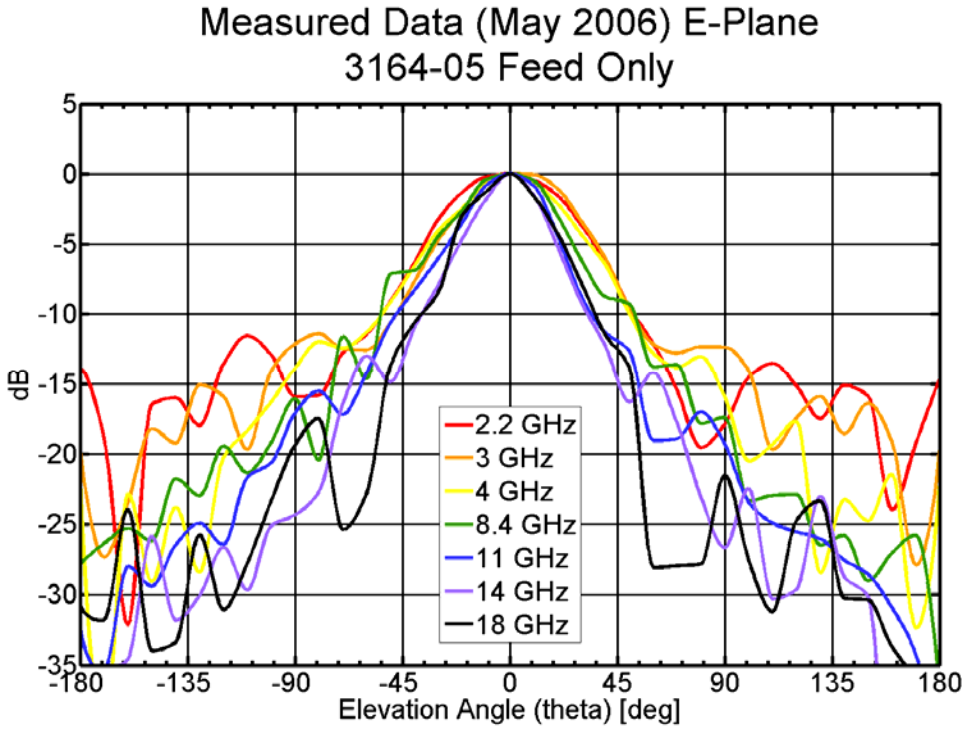
1.1.2 Return Loss

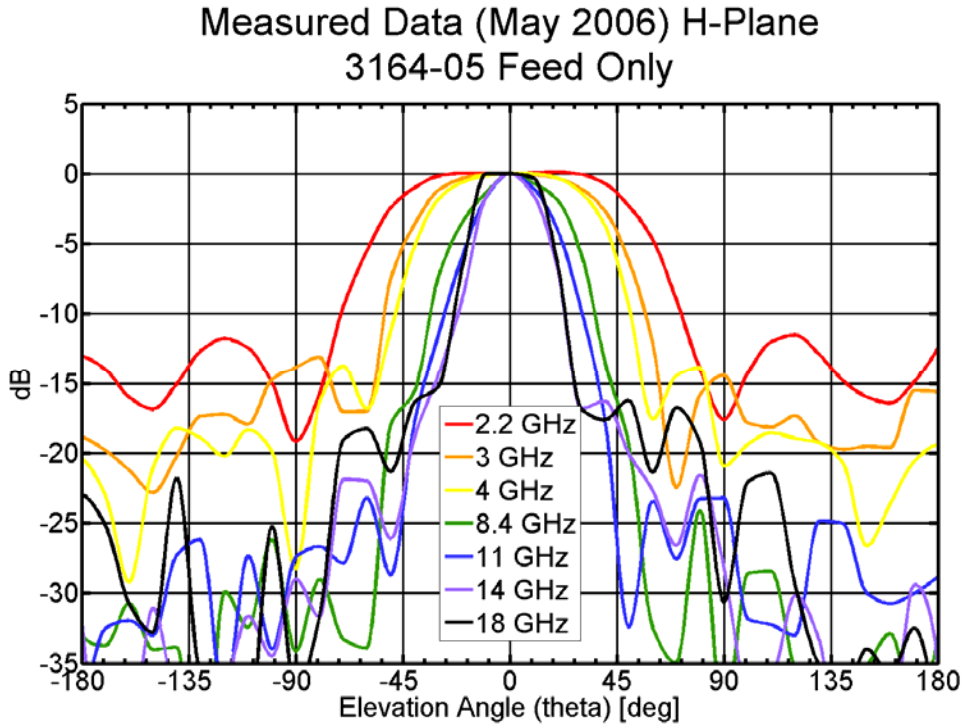
The following return loss data was measured by Hamdi Mani and Sander Weinreb on April 27, 2006.



1.1.3 Patterns for Feed Only

The following far-field pattern data was measured by Hamdi Mani and Sander Weinreb in May 2006.



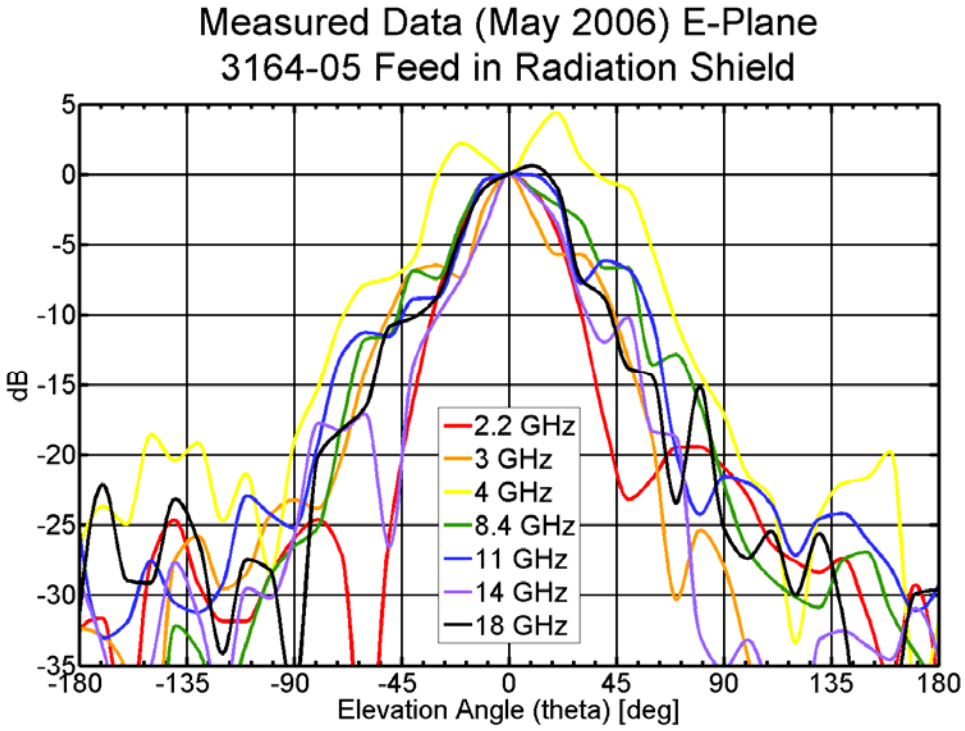


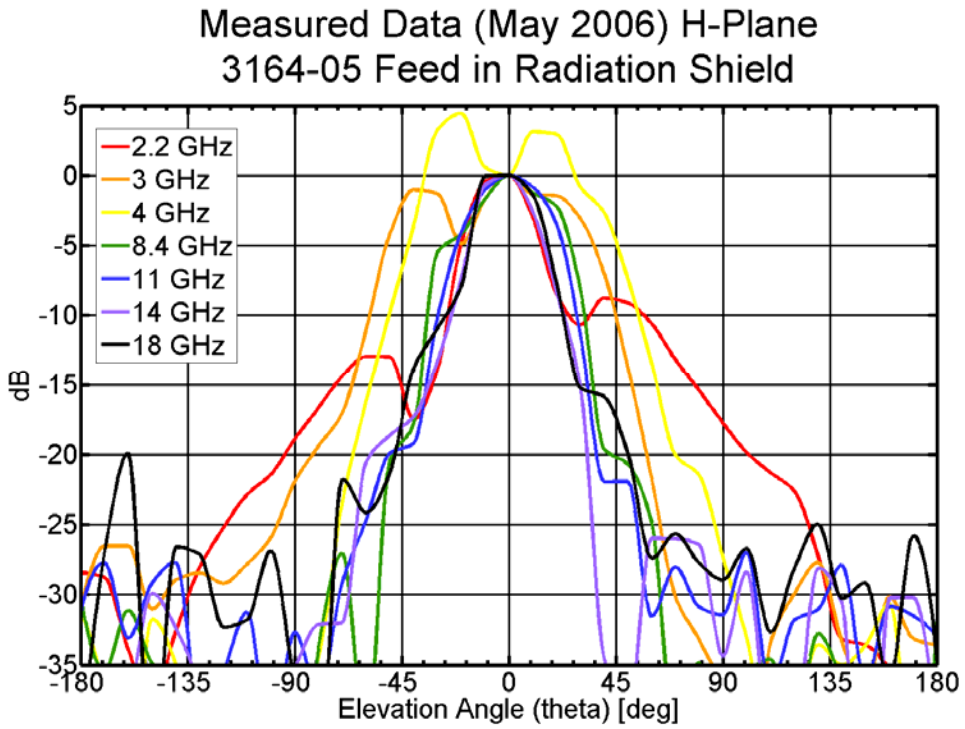
1.1.4 Patterns for Feed in Radiation Shield

In order to improve the far-field patterns, the feed was placed in a radiation shield. The following image shows the feed inside the radiation shield. In addition to radiation shield, there is a metal back-plate (not shown).

Figure 1.1.4 a

The following far-field pattern data was measured by Hamdi Mani and Sander Weinreb in May 2006.





1.1.5 Patterns for Feed in Absorber Lined Radiation Shield

Far-field patterns were further improved by lining the radiation shield with AN-73 microwave absorber. The following image shows the feed inside the absorber lined radiation shield. The absorber was put on the sides of the radiation shield as well as the base of the back-plate.

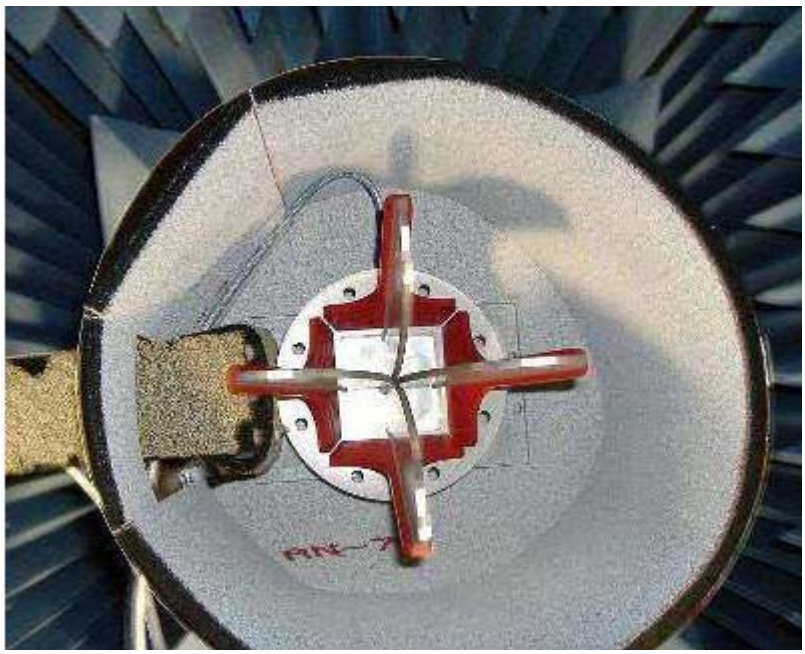
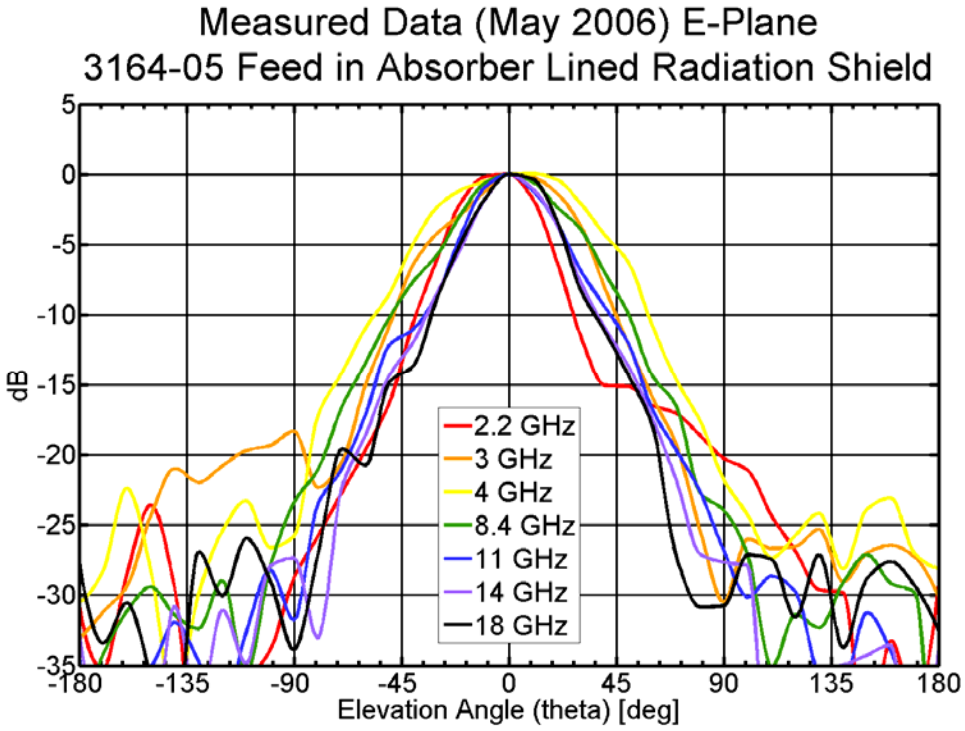
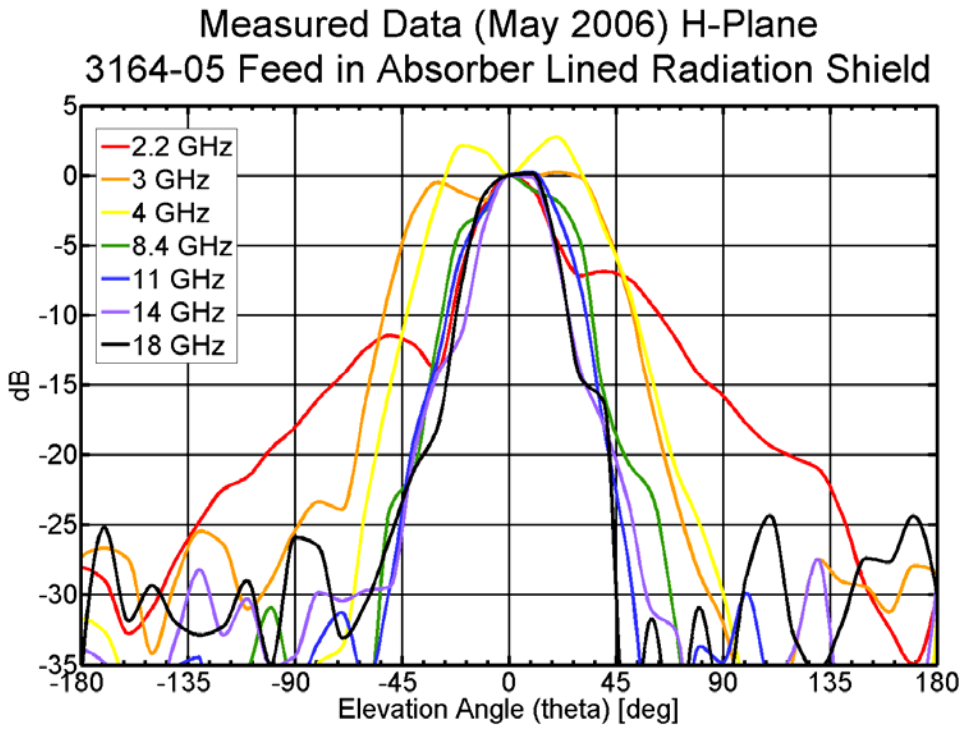


Figure 1.1.5 a

The following far-field pattern data was measured by Hamdi Mani and Sander Weinreb on May 18, 2006.





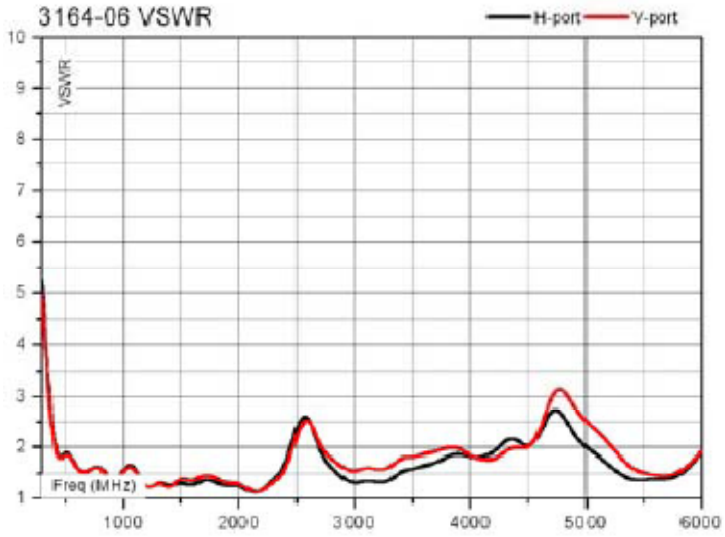
1.2 ETS-Lindgren 3164-06 (0.3-6 GHz)

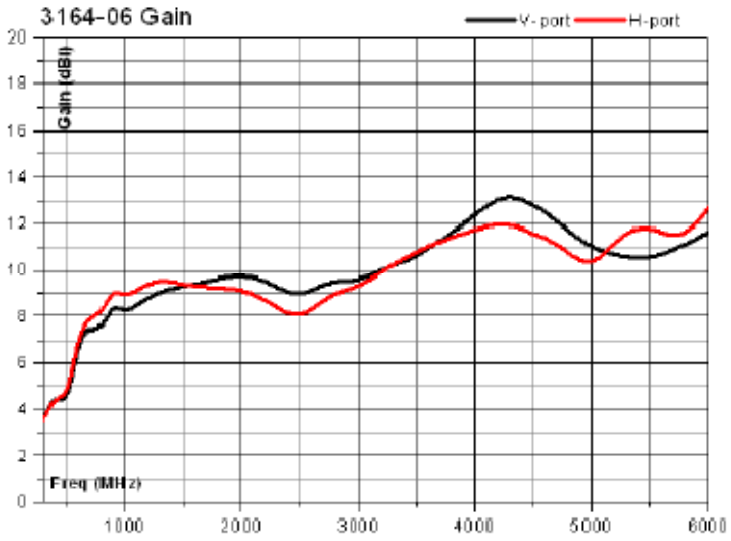
The ETS-Lindgren 3164-06 is 3x scaled variation of the 3164-05 quad-ridge horn feed designed by V. Rodriguez [1-1]. It has an operation frequency of 300 MHz - 6 GHz. It will be used as the feed for the lower frequency range of the Goldstone Apple Valley Radio Telescope (GAVRT) [1-2]. The entire receiver system is described in [1-3]. The following image shows the 3164-06 feed.

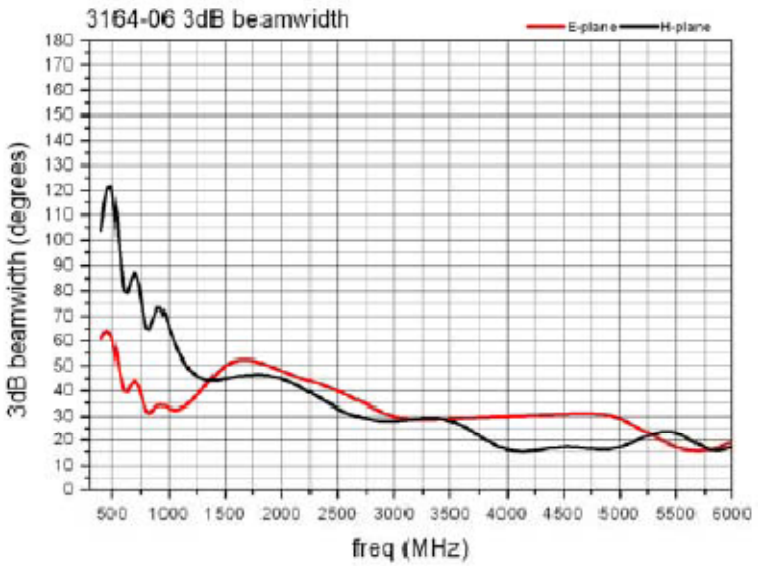
Figure 1.2 a

1.2.1 Data Sheet

The following plots are from the ETS-Lindgren 3164-06 data sheet [1-6]:



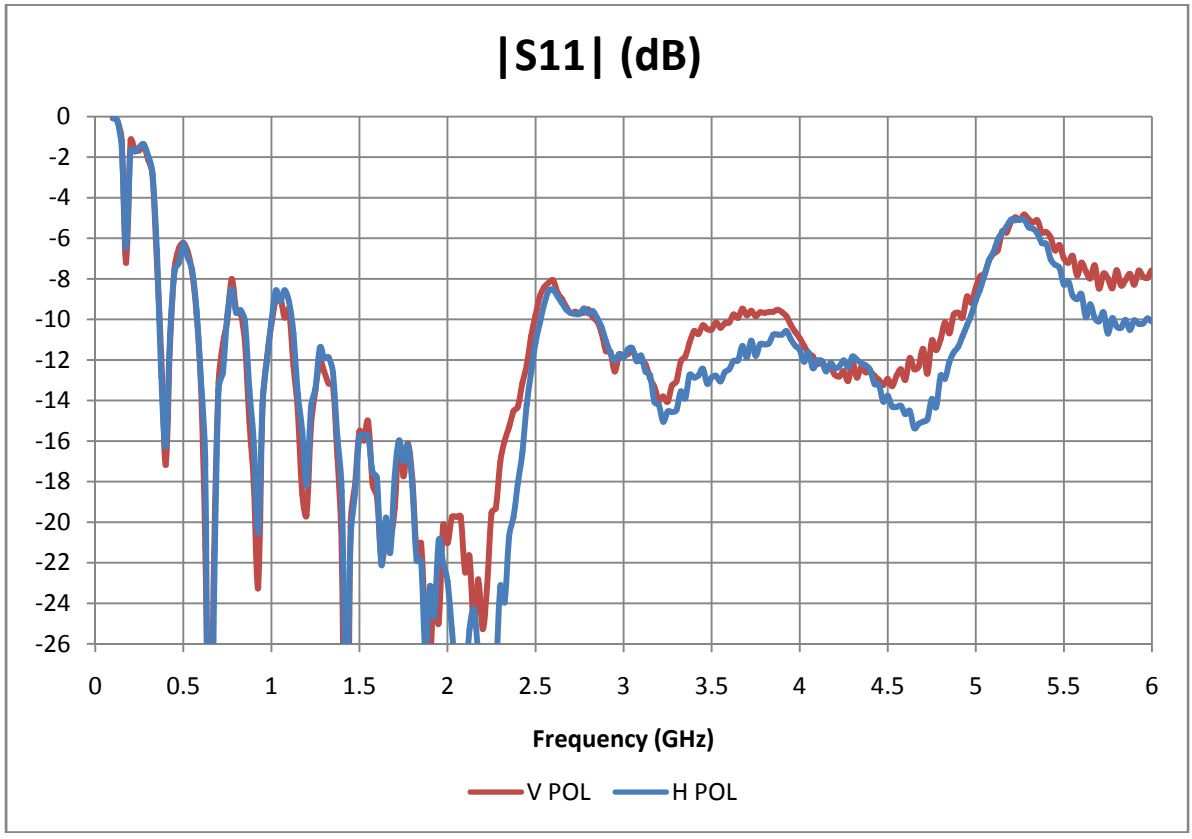




For more information, please see the user's manual [1-5].

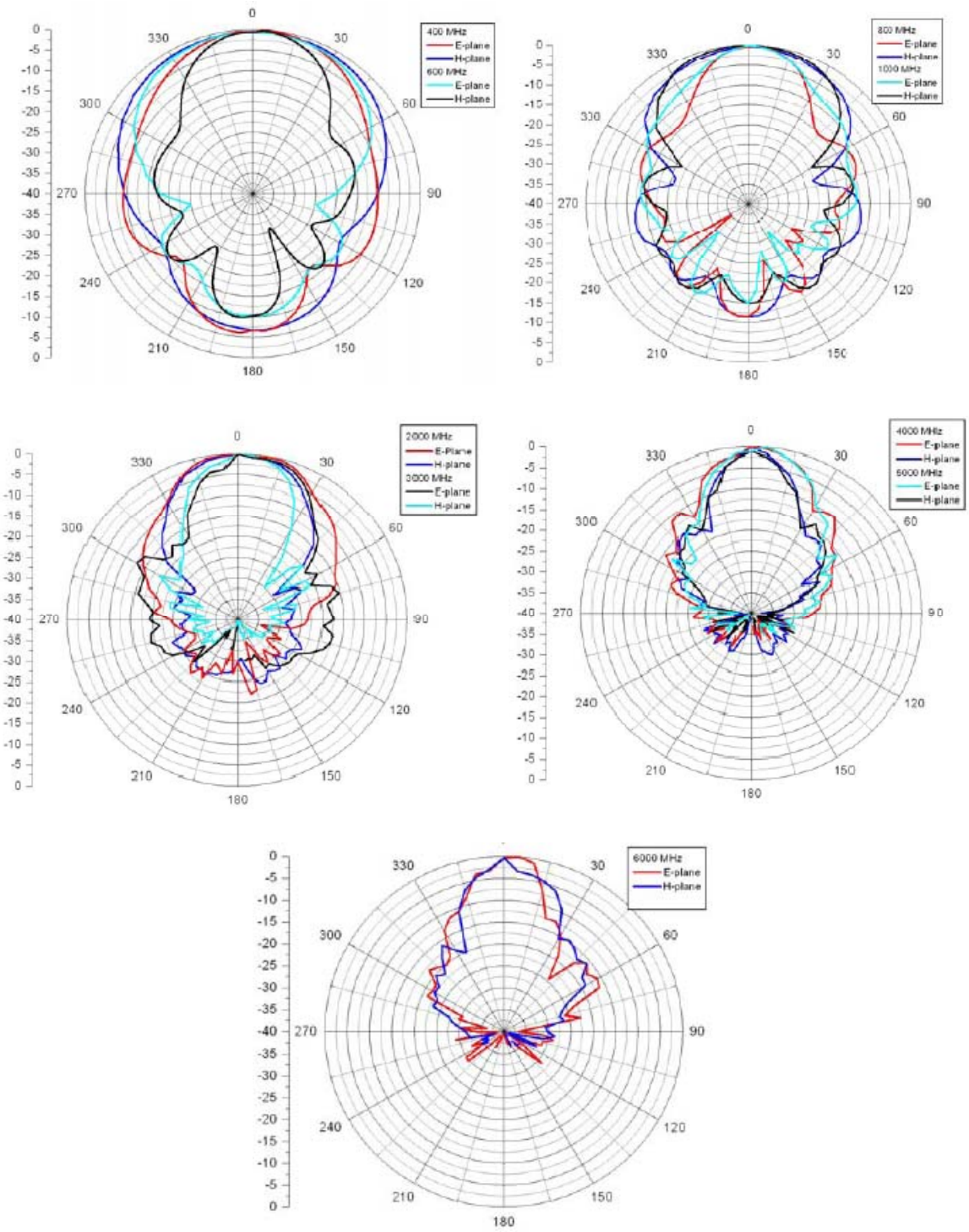
1.1.2 Return Loss

The following return loss data was measured by Hamdi Mani and Sander Weinreb on July 31, 2007. The feed came with a lossy backshort, which was later removed to improve the noise of the feed. However, the data below is for the complete feed, with the lossy backshort.



1.2.3 Patterns from Data Sheet

The following far-field patterns are from the ETS-Lindgren 3164-06 data sheet [1-6].



2

Modeling Software

2.1	QuickWave	19
2.1.1	Features	19
2.1.2	Basic Usage Information	19
2.1.3	Problems, Difficulties, and Errors	22
2.2	CST Microwave Studio	23
2.2.1	Features	23
2.2.2	Basic Usage Information	23
2.2.3	Problems, Difficulties, and Errors	26
2.3	Comparison	26
2.3.1	QuickWave Advantages	27
2.3.2	CST Advantages	27
2.3.3	Conclusions	27

The modeling and simulation of the Lindgren feeds has been done using two separate electromagnetic simulation suites, QuickWave and CST Microwave Studio. Each of these software packages is described in detail in this section. In addition, a brief comparison of the two software packages is made in the final sub-section.

2.1 QuickWave

Information about QWED QuickWave version 6.5.

2.1.1 Features

The QuickWave package is developed by QWED out of Warsaw, Poland. QuickWave uses a conformal polygon mesh and a finite difference time domain (FDTD) solver. The software package has two main programs, which are completely separate. The QW-Editor allows the user to define the structure and its mesh. Then QW-Simulator is run to perform the electromagnetic simulation on the structure and produce all outputs. Below are the descriptions of QW-Editor and QW-Simulator from the QWED webpage [2-1].

QW-Editor:

> A graphical editor for definition of geometry, media, I/O parameters and postprocessing. It comprises a library of parameterized objects and a capability for generating further objects and libraries. Conversion to and from CAD formats is also facilitated.

QW-Simulator:

> A unique, conformal FDTD solver. Its output data include multi-modal, multi-port S-matrices, radiation and scattering patterns, pattern of field, dissipated power, time-domain reflectometry etc.

2.1.2 Basic Usage Information

There are two ways to define a structure in QW-Editor. It can be done manually, using the graphical user interface, or code can be written in the QuickWave user defined object (UDO) language. I highly recommend creating structures using UDO code, because it allows changes to be made much easier. Using a UDO script, you can define several variables and then create your structure based on those variables. In the editor, a user can simply change a variable, and then the whole script is re-run with the new values. Without using a UDO script, there is no way to change certain parameters of your structure without re-drawing everything. The UDO language is described in detail in the QW-Editor Reference Manual [2-2]. In addition, the UDO code that I wrote to create the model of the Lindgren feed is discussed in section 3.3.

An important feature to understand is the concept of bi-phased (BP) objects. The QuickWave editor has two modes, draft mode and final mode. In draft mode, BP objects are simplified and are usually just drawn to show their extent, but not the actual shape. This is necessary, because BP objects are directly connected with the mesh. For example, and sphere made up of slices, where each slice is the thickness of one mesh cell. Clearly, this object cannot be defined before the mesh. In addition, mesh changes need to occur before the object is created. QuickWave uses BP object to handle these problems. When in draft mode, the mesh does not need to be defined, and the object is drawn only roughly. Once the mesh has been created, the user can switch to final mode, and the BP object is redrawn based on the confines of the defined mesh. Changes can be made to mesh by switching back to draft mode. After making the change, switching to the final phase will re-draw the object. This method is highly recommended in order to create

structures that act they way that you intend. It allows you to control exactly how the structure and the mesh are related. This can only be done using scripts defined in the UDO language. The script used to create the Lindgren feed (discussed in section 3.3) is such an example.

I will give a brief overview of the tasks that you will want to perform in order to run a basic simulation. Most of this can be done inside of a UDO script, as I have done in the script described in section 3.3.3.

1. Define the background medium
   Parameters>Circuit
   Select either Metal or Air
2. Define all materials
   Parameters>Materials
   Create a dielectric medium, and define the material properties. The material properties are defined as a real epsilon and mu in addition to an electric and magnetic conductivity (sigma).
3. Define the units
   Parameters>Units
4. Define the structure in draft mode
   Use the buttons at the top of one of the 2D QW View windows
   Make sure the structure includes at least one port
   In order to compute far-field patterns, the entire structure needs to be enclosed in a near-to-far (NTF) box. The NTF box object is included with QW in elib/boxes
5. Define the mesh
   Open the mesh parameters dialog by pressing the button with the blue grid at the top of any 2D QW View window
   Input the maximum cell size for all three directions. The maximum cell size will be the dimensions of the largest cell in your mesh. All of the cells will be this size unless your structure geometry or mesh snapping planes force a refined or modified mesh
   You can check the size of your mesh by clicking on the button with the lowercase ‘i’ over a blue grid.
6. Change to final mode
   Project>Final Modify All
   Switching to final mode will draw all of the BP objects in their final form, which is based on the predefined mesh.
7. Configure excitation
   Parameters>I/O Ports
   Select the port name from the pull-down menu at the top left
   Select the port parameters such as exciting field and impedance
   Choose the excitation waveform and frequency range on the right side of the dialog window
8. Configure the simulation postprocessing outputs
   Parameters>Postprocessing
   Select the postprocessing outputs that you would like

Select [S]-differential with a frequency range and step size to obtain S-parameters results
Select Near-to-Far in order to obtain far-field patterns. You must enter a space delimited list of the frequencies from which you would like to obtain far-field patterns

9. Determine when to stop the simulation
   Open Amigo using Parameters>Amigo
   Near the bottom, there is a field that lists the number of iterations required. If this value is unreasonably high, then you will need to adjust your mesh. This value is influenced greatly by the size of the smallest cell. If this value is too large, then change your mesh to increase the size of the smallest cell. Remember to change to draft mode before changing your mesh, and then change back to final mode. Remember this value for the maximum number of iterations.
10. Export project to QW-Simulator
    File>Export & Run
    This will open QW-Simulator with your current project open
    At this point you can define breakpoints by using Configure>Breakpoints. Breakpoints allow you to automate certain tasks while running the simulator. For example, you can automatically set to simulator to stop after a certain number of iterations, or save far-field patterns at a specified point in the simulation. Setting breakpoints is described in more detail in section 4.2.1.
11. Run the simulator
    Run>Start
    The simulator will do some initial preprocessing, and then the simulation will start. While the simulation is running, the number of iterations is displayed in the lower right corner of the screen.
    The QW simulator will run forever unless you stop it, or have it set to stop using a breakpoint. You will want to run the simulator for at least as many iterations as Amigo estimated. I tend to double that number to be sure that the values have converged. While the simulator is running, S-parameter results can be viewed as well as the instantaneous fields in the structure. All of these results can be viewed from the View menu. Getting outputs from the simulator is discussed in much more detail in section 4.
12. Save the simulation
    When you have completed your simulation, you can suspend it using Run>Suspend.
    At this point, I recommend saving all of the simulation values stored in memory using File>Freeze. This saves the current state of the simulator, and allows you to resume at this same point or obtain more results at a later time.
13. View and save simulation results
    See section 4 for instructions on getting results from the simulation.

This is a basic overview of how to do common tasks in QuickWave. For more details, please consult the reference manuals [2-2] [2-3].

2.1.3 Problems, Difficulties, and Errors

I have had many difficulties using QuickWave. I will list some of my opinions about the problems that I have had.

QuickWave is not a very user-friendly software package, and therefore it takes a very long time to become accustomed to its idiosyncrasies. The user's guide [2-4] can be helpful; however, it is poorly organized and difficult to obtain useful information from it. It is basically just a tutorial and not a user's guide. The only way to learn how to use the software is to go through the tutorial, and try to figure out how they are doing things. I found it more useful to read through the reference guides [2-2] [2-3], which describe the software in detail.

One major disadvantage of QuickWave is that the drawing is very closely connected with the mesh, and as a result, they must be modified together. It is almost impossible to make a change to the mesh without changing the structure, and vice versa. It is much more desirable to have a defined structure, and then modify the mesh to accurately represent the structure. As a result of this interdependence between the mesh and the structure, I think that good structures can only be defined using UDO language code. As a result of this, it takes a very long time to build a model in QuickWave. In addition, it becomes very difficult to make changes to the structure. For an example of these difficulties, see section 3.3.1.

The software is also very unstable and will crash often. It becomes very frustrating to encounter a segmentation fault every 15 minutes! In addition, many of the features of the user interface do not work as expected (or at all). I found myself spending the majority of my time finding workarounds to bugs in the software than doing anything else. The software seems to have bad memory allocation, and usually crashes due to some sort of memory error. It also appears to have several memory leaks, which cause the system memory to fill up, which then leads to more problems. One of the most frustrating problems occurs when trying to export a large structure. The QW-Editor will halt with the following error message:

> Out of memory! Cannot reallocate ####### descriptors (312 bytes each)
> Try to change CellDescrChunk (MeshParams section in zednqw.ini file)
> I:\qed\qed65\qed\qedmesh.cpp line=847

It is very strange that the program reports being out of memory, because I was certainly not out of memory. In fact, less than 30% of my memory was in use. I have contacted QW support about this problem on numerous occasions, but they do not seem to know what is causing it, although they have experienced it themselves. They believe that it is an error in the Windows memory manager. They have been able to work around the problem with limited success by changing the value of the variable CellDescrChunk in the file zednqw.ini. However, they do not know if the value should be increased or decreased, they just told me to play with it until I am able to export the structure. I have not had much success with working around this problem. I was able to get a test structure to export by changing that value on a different computer than my primary machine. However, I have never been able to export the desired structure. As a result of this error, I am not able to simulate my model of the Lindgren feed inside of a radiation shield at the higher frequencies of operation.

I have e-mailed the support personnel at QuickWave on several occasions, but they have not been extremely helpful. There is a about a one day delay in getting a response from them; however this is due mostly to the time difference between Los Angeles and Warsaw. When they do respond, they are usually able to tell me more about the problem that I am having, but can rarely actually fix the issue.

2.2 CST Microwave Studio

Information about CST Microwave Studio 2006B.02, Educational Version.

2.2.1 Features

CST Microwave Studio (MWS) is part of the CST Studio Suite that is developed by Computer Simulation Technologies (CST GmbH) out of Darmstadt, Germany. Throughout this document, CST Microwave Studio will be referred to by simply CST. CST contains three separate solvers: Transient (Time Domain) Solver, Frequency Domain Solver, and an Eigenmode Solver. In addition, CST supports two mesh types, hexagonal and tetrahedral. In addition, there are many automated meshing utilities, including adaptive subgridding. CST has a very user friendly UI which makes using the software very straightforward. Below is the description of Microwave Studio from the CST webpage [2-5].

> CST MICROWAVE STUDIO® (CST MWS) is a specialist tool for the fast and accurate 3D EM simulation of high frequency problems.
> Applications include the expanding areas of: Mobile Communication, Wireless Design (bluetooth), Signal Integrity, and EMC. Exceptionally user friendly, CST MWS quickly gives you an insight into the EM behaviour of your high frequency designs.
>
> Beside the flagship module, the broadly applicable Time Domain solver and the Frequency Domain solver which simulates on hexahedral as well as on tetrahedral grids, CST MWS offers further solver modules for specific applications. Filters for the import of specific CAD files and the extraction of SPICE parameters enhance design possibilities and save time. In addition, CST MWS is embedded in a larger design environment through CST DESIGN STUDIO™ whose open architecture provides a link with external simulators.

This document only focuses on a few features of CST MWS including the Transient Solver using a hexagonal mesh. For more information about CST, please refer to the manuals [2-6] [2-7].

2.2.2 Basic Usage Information

CST Microwave Studio has a very simple and intuitive graphical user interface. Like QuickWave, most tasks can be preformed manually or using a pre-programmed script. However, in CST, most tasks can be done manually very easily and therefore there is no need to write any code. However, it can be useful to write some macros to make it easier to perform common and time-consuming tasks. CST macros must be written in Visual Basic for Applications (VBA). CST comes with good documentation on the language, and it is very simple to write basic

commands. Additionally, all commands in CST use the VBA language. Even when using the graphical user interface, CST is converting your key presses into VBA code that it will run when you click Update. This code is viewable and can be modifying the History List (see below for details). For more information on the VBA language, refer to the Help menu in CST MWS.

A very nice feature of CST MWS is that the drawing is completely separate from the mesh, and therefore each can be modified separately. This means that the drawing can be easily imported from another drawing program. See the CST manual [2-6] for more information about importing external formats. This also means that you can draw the actual structure instead of drawing a simplified model that is easier to mesh. Once the drawing is complete, you can then define the mesh to be sure that the structure is being meshed correctly. Making changes to the structure automatically changes the mesh if necessary, however changing the mesh does not require you to change your structure.

An important feature to understand is the History list. All actions taken by the user are recorded in the history list. Every time you click Update, CST will run all commands in the history list in order from top to bottom. Therefore, changing the value of a variable will become a global change once the structure is updated because at this point all past actions will be re-evaluated using the new values of all variables. The history list is also very import for making changes to the structure, because this history can be modified. To open the history list, click Edit>History list. Any item in the list can be modified by clicking the Edit button. In addition, you can restore the state of program to a "past time" by using the Restore button. This can allow you to add events at a previous time by restoring to some point and then defining new structures. At any point in the history list, the program will only recognize objects created at a previous time. For example, say you are at the bottom of the history list and add a new material. Then edit the definition of a brick at some point higher on the list to be made of the new material. This is an invalid action, because the program will not be aware of the new material until it reaches the end of the list. You can get around this by restoring the history to some point before the definition of the brick and then defining the new material. Now it will work, because when the brick is created the material will already be defined.

I will give a brief overview of the tasks that you will want to perform in order to run a basic simulation.

1. Define the units
   Solve>Units
2. Define the background medium
   Solve>Background Material
   For air select Normal, Epsilon=1, Mue=1
3. Define all materials
   Solve>Materials>New Material
   This can be done at any time, but it is nice to do it at the beginning
   To create a dielectric medium, select type=Normal and define the material properties. The material properties are defined as a real epsilon and mue in addition to an electric and magnetic sigma (on the Conductivity tab).

4. Define the frequency range
   Solve>Frequency
   This will define the bandwidth of the excitation pulse in addition to the frequency range of S-parameter analysis
5. Define the boundary conditions
   Solve>Boundary Conditions
   For far-field pattern analysis, you will want open space all around the structure, perhaps with the exception of a ground plane. For far-field analysis, the mesh requires some empty cells outside of the structure. If you select, 'open (add space)' CST will automatically insert the correct number of extra cells.
6. Define all variables
   At the bottom of the screen, there is a table with the tab labeled 'Global.' These are the global variables. This allows you to assign named variables to certain quantities. Variable names or expressions involving variable names can be entered into any value field in the program. This allows you to completely parameterize your structure. It is very easy to change the values of the variables and then apply the changes to the structure.
7. Define the structure
   This can be done by importing an external design file, or defining all structures manually. Structures can be defined manually by creating one the basic shapes: Objects>Basic Shapes. You can select the size and placement of the shape with the mouse, or using values by typing <Esc>. Remember that the dimensions can be expressions involving variable names. Be sure to include at least one port. This can be a waveguide port, discrete port, or plane wave. All can be added using the Solve menu.
8. Check the mesh
   Click Mesh>Mesh View to enter the mesh viewing mode. You can step through the mesh using the increment and decrement index options from the Mesh menu. This allows you to visually inspect the entire mesh. At the bottom right corner of the screen, there is a cell that tells the number of mesh cells. As you modify the mesh, make sure that this value is reasonable. Click Mesh>Global Mesh Properties to modify the mesh. Lines per wavelength is an important value which allows you to define the maximum cell size in term of lines per wavelength (at the highest frequency). Other important controls can be found in the Specials submenu. On the Refinement tab, you can control the mesh refinement at material edges and inside lossy materials. On the Advanced tab, you can control extra features such as TST cells and subgridding. Note: subgridding can reduce the total number of cells, but it takes a long time and a large amount of computer resources. It is only worthwhile if it improves the number of mesh cells by a factor of three or more. The mesh properties can also be defined for individual object by right clicking on the object name from the list at the left and clicking Mesh Properties.
9. Configure simulation outputs
   Select the desired outputs from the Solve menu. There are several types of outputs that are grouped into field monitors, voltage monitors, and probes.

In order to obtain far-field patterns, select Solve>Field Monitors, and then select Farfield/RCS for a given frequency.

10. Run the solver

    Solver>Transient Solver

    You can select the accuracy. The solver will stop when this much energy remains in the system. You can start with -30 dB, and then decrease the value as you become more confident in your mesh, and would like very accurate data.

    First, the solver will do some initial preprocessing, and then the simulation will start. While the simulation is running, several of the selected outputs can be viewed. Clicking on 1D Results/|S| dB will show the S-parameters for the defined S-parameter ports. 1D Results/Energy will show a plot of the energy in the structure. This will give you an idea of how close the solver is to finishing

11. View and save simulation results

    See section 4 for instructions on getting results from the simulation.

This is a basic overview of how to do common tasks in CST Microwave Studio. For more details, please consult the reference manuals [2-6] [2-7] in addition to the built-in Help.

2.2.3 Problems, Difficulties, and Errors

I have not found very many difficulties using CST; however, I will list the one issue that I have had.

The feature of adaptive subgridding has not been very useful to me. It appears to take a very long time and an extreme amount of physical memory in order to run. I have seen it take several hours to mesh a structure that would run in about 1 hour to run without subgridding. In addition, it is often unable to create the mesh due to a lack of memory (I had 4 GB available). After running out of memory, the program crashed. However, this is understandable since it actually was out of memory. CST warns that subgridding is not advantageous unless it is able to reduce the number of cells by a factor of three. I have not seen this much improvement, and it has certainly not been faster for my structures. I would therefore recommend not using subgridding unless it dramatically decreases the overall time of simulation.

2.3 Comparison

Both QuickWave and CST MWS take about the same amount of time to run the simulation. They solve the same structure of the full Lindgren 3164-05 feed between 1 to 3 hours. However, other time intensive steps other than the simulation time separate them. In addition, there is a major difference between them in terms of ease of use, stability, documentation, and features. The following table summarizes the time comparison between the two software packages:

Table 2.3 a

Time to…	**QuickWave**	**CST MWS**
Become familiar with software	Weeks	Days
Create a new structure	Days	1 hour
Modify the mesh	<10 min	Seconds (hours when using subgridding)
Run full simulation of Lindgren 3164-05 Feed	1-2 hrs.	1-3 hrs.
Export far-field patterns over 1-18 GHz in 0.5 GHz steps, with 1° theta resolution and 5° phi resolution	10 hrs.	50 hrs.
Total time to run a new model (excluding becoming familiar with software)	**3 days**	**3 days**

It is important to realize that although both software packages appear to take about 3 days to run a new model, these times are very different. In the case of QuickWave, the majority of this time requires the user to be actively working on the project. In contrast, the majority of the time running CST is spent exporting far-field patterns. This is an automated process and requires absolutely no user intervention. Therefore, it takes much less human resource time to use CST MWS.

2.3.1 QuickWave Advantages

- Significantly faster postprocessing and extraction of far-field patterns
- Gives the user more control over the meshing of the structure

2.3.2 CST Advantages

- Very simple and powerful user interface with good documentation
- Very stable
- The structure drawing is completely separate from the mesh and therefore each can be modified independently
- Modifications are very easy to make, and a built-in optimizer can even sweep certain parameters to determine an optimal value
- Computes many statistics including the phase center
- Can simulate all structures, no matter how large (as long as it has enough resources)

2.3.3 Conclusions

I believe that CST MWS is a much better software package than QuickWave, and therefore I highly recommend using only CST to model structures similar to the Lindgren wideband quad-ridge slot feeds. It is not a bad idea to run the model in both CST and QuickWave and then compare; however, I do not think that it is worth the time to get the QuickWave model to work well. As a result, most of the data in this report is from CST simulations.

3

CAD Models of Lindgren Feeds

3.1	Mechanical Drawings and Measurements	29
3.1.1	Measurements from 3164-05 Inventor File	29
3.1.2	3164-05 Physical Measurements	33
3.1.3	3164-06 Physical Measurements	37
3.2	Modeling Microwave Absorber	41
3.3	QuickWave Model	42
3.3.1	How to build the model	42
3.3.2	Scripts to create QuickWave UDO code	43
3.3.3	QuickWave UDO Code	57
3.4	CST Model	96
3.4.1	How to build the model	96

This section describes the CAD models of three different structures. It describes the mechanical measurements of all three structures, and how each of the structures is defined in both QuickWave and CST Microwave Studio. This section all includes all details for implementing these designs in QwickWave or CST. In addition, there is a sub-section to discuss the modeling of microwave absorber material.

3.1 Mechanical Drawings and Measurements

This document includes CAD models from three different structures.

The first structure is called the 3164-05 Inventor file design. It is a very similar design to that of the actual ETS-Lindgren 3164-05 feed; however, it is slightly different. This design came from a set of AutoDesk Inventor files that were sent to us from the manufacturer, ETS-Lindgren.

The second structure is the actual ETS-Lindgren 3164-05 feed (2-18 GHz). It was created based on mechanical measurements, which were made from the feed itself.

The third structure is the actual ETS-Lindgren 3164-06 feed (0.3-6 GHz). It was created based on mechanical measurements, which were made from the feed itself.

All measurements are in mm, unless noted otherwise.

3.1.1 Measurements from 3164-05 Inventor File

All measurements were made from the AutoDesk Inventor file. The shape of the fins in the Inventor files is different from that of the actual feed. This affects the far-field patterns particularly at the lower frequencies. The image below shows the shape of the fin described in the Inventor files.

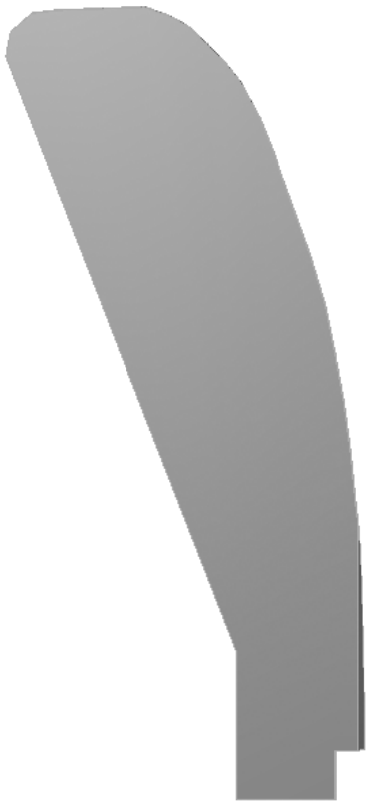


Figure 3.1.1 a

Figure 3.1.1 b shows the shape of the beveled edge of the fin inside the slot. In addition, the measurements used to model the beveled edge are shown.

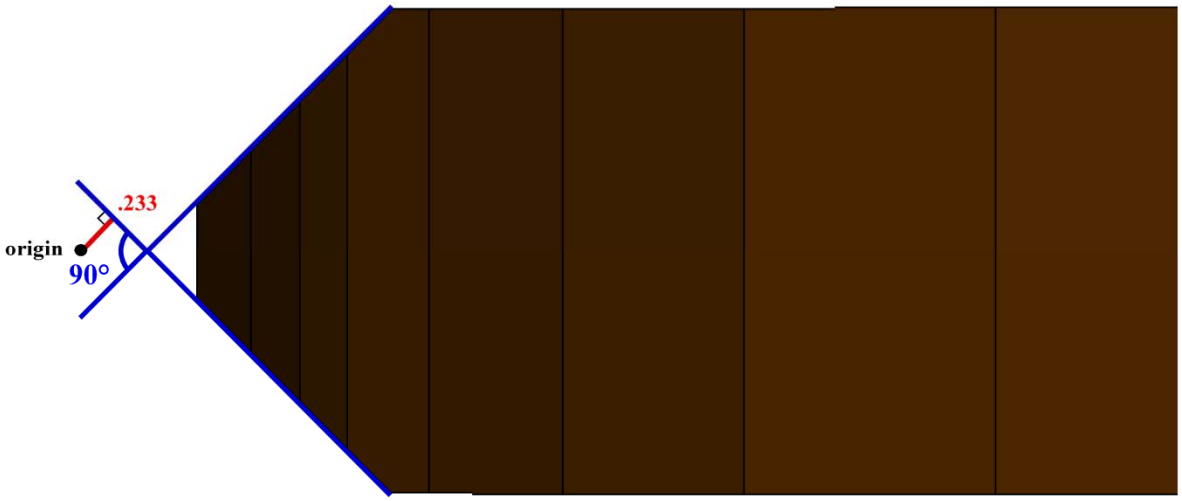


Figure 3.1.1 b

Here is a summary of the measurements. These exact variables are used to make the CST model. These values were also used in the UDO script that creates the QuickWave model (see section 3.3.3).

Table 3.1.1 a

Name	Value (mm)	Description
gap_w	0.686*2	width of the gap between opposing fins
fin_x_shift	gap_w/2	x position to shift the fin coordinate origin
fin_z_shift	-0.228	z position to shift the fin coordinate origin
fin_thick	3.615	thickness of the fins
base_h	33.726	outer height of the base box, not including the base_top
base_w	60.249	outer width of the base box
base_z	-2.502	z position of the bottom of the inside cutout of the base box
cut_depth	22.581	depth of the inside cutout of the base box
cut_w	53.889	width of the inside cutout of the base box
hemi_r	13.02/2	radius of the hemisphere cutout in the base box
base_top_dia	101.6	diameter of circle that makes up the top of the base
base_top_thick	3.175	thickness of the circle that makes up the top of the base
base_top_cutout	60.249	side length of the square cutout in the circle that makes up the top of the base

For the model that includes the radiation shield and absorber, here are the values used in the CST model.

Table 3.1.1 b

Name	Value (mm)	Description
feed_top	152.451	z position of the top of the feed by itself
feed_bot	base_z-(base_h-cut_depth)	z position of the bottom of the feed by itself
shld_ir	115-shld_thick	inner radius of the radiation shield
shld_thick	0.864	thickness of the sides of the radiation shield
shld_base_thick	6	thickness of the base plate under the feed
abs_thick	10	thickness of the absorber material

The shape of the fin is defined differently in QuickWave and CST. One way to describe the shape of the fin to stack differently sized trapezoids vertically. This form is used to create the QuickWave model (see section 3.3). The second method is to list the points that draw out the curve of the fin. This is the way that the CST model makes the shape of the fin (see section 3.4). In this section, the fins are described using both forms.

In QuickWave, the fin shape is defined by a series of trapezoids that are stacked vertically. The table below defines each of the trapezoids. Section 3.3.2 describes how this table is converted to the UDO code that creates the QuickWave model. The origin of the coordinate system is center of the gap at the z-position of the excitation. For each trapezoid (sector), the top (top_z) and bottom (bot_z) planes are defined. In addition, the four corners are defined: the front and back edges for the top and the bottom. The angle of the front and back edges is also specified to allow QuickWave to break these large sectors into many smaller sectors (see section 3.3.3).

Table 3.1.1 c

sector	bot_z	bot_fx	bot_bx		top_z	top_fx	top_bx		f_angle	b_angle
1	-10.472	3.510	26.949		-0.228	3.510	26.949		90	90
2b	-0.228	0.686	26.949		9.906	0.686	26.949		90	90
3b	9.906	0.686	26.949		20.079	0.896	26.949		88.82	90
4b	20.079	0.896	26.949		30.251	1.092	30.884		88.82	68.85
5b	30.251	1.092	30.884		40.437	1.448	34.825		88.05	68.85
6b	40.437	1.448	34.825		45.415	1.803	36.751		85.89	68.85
7b	45.415	1.803	36.751		50.622	2.413	38.765		83.34	68.85
8b	50.622	2.413	38.765		60.808	3.404	42.706		84.52	68.85
9b	60.808	3.404	42.706		70.993	4.750	46.646		82.37	68.85
10	70.993	4.750	46.646		81.178	6.604	50.586		79.79	68.85
11	81.178	6.604	50.586		91.364	8.636	54.527		78.69	68.85
12	91.364	8.636	54.527		101.549	11.684	58.467		73.31	68.85
13	101.549	11.684	58.467		121.920	19.025	66.348		70.2	68.85
14	121.920	19.025	66.348		129.540	22.073	69.296		68.15	68.85
15	129.540	22.073	69.296		137.185	26.137	72.254		61.99	68.85
16	137.185	26.137	72.254		140.995	29.185	73.728		51.34	68.85
17	140.995	29.185	73.728		142.265	30.405	74.219		46.16	68.85
18	142.265	30.405	74.219		144.831	32.868	73.812		46.16	99
19	144.831	32.868	73.812		147.371	35.578	73.279		43.15	101.86
20	147.371	35.578	73.279		148.641	36.932	71.831		43.15	138.74
21	148.641	36.932	71.831		150.546	40.386	69.660		28.88	138.74
22	150.546	40.386	69.660		151.181	42.130	68.936		20.01	138.74
23	151.181	42.130	68.936		151.816	43.873	63.525		20.01	173.31
24	151.816	43.873	63.525		152.451	45.617	45.617		20.01	177.97

In CST, the fin shape is defined by a set of points that trace out a curve. The following points define this curve. The origin of the coordinate system is the edge of the fin at the bottom of the gap.

Table 3.1.1 d

point number	z position	x position
0	0	0
1	10.134	0
2	30.479	0.406
3	40.665	0.762
4	45.643	1.117
5	50.850	1.727
6	61.036	2.718
7	71.221	4.064
8	81.406	5.918
9	91.592	7.950
10	101.777	10.998
11	122.148	18.339
12	129.768	21.387
13	137.413	25.451
14	141.223	28.499
15	145.059	32.182
16	148.869	36.246
17	150.774	39.700
18	152.679	44.9311
19	152.044	62.839
20	151.409	68.250
21	147.599	72.593
22	145.059	73.126
23	142.493	73.533
24	20.307	26.263
25	-10.244	26.263
26	-10.244	2.824
27	0	2.824
28	0	0

3.1.2 3164-05 Physical Measurements

Almost all measurements were made with calipers, with the exception of the fin shape. The fin shape was determined my tracing its outline on graph paper, and then making measurements from the image. The following sheets are my original notes from measuring the 3164-05 feed.

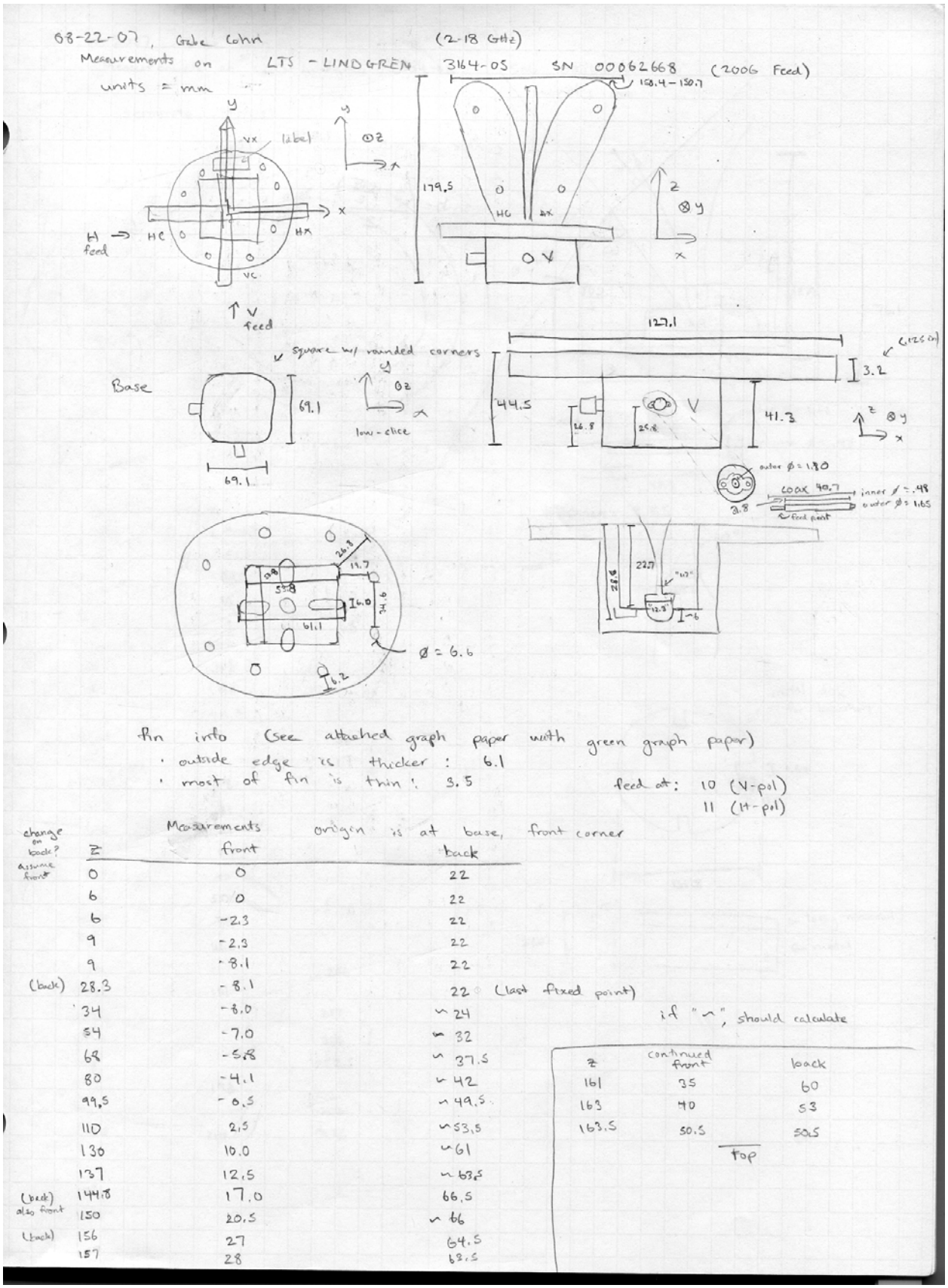

08-22-07, Gabe Cohn (2-18 GHz)
Measurements on LTS - LINDGREN 3164-05 SN 00062668 (2006 Feed)
units = mm
label
VX
HC
HX
VC
H feed
V feed
158.4 - 158.7
179.5
HG
VX
V
square w/ rounded corners
Base
69.1
69.1
low-chce
127.1
3.2
44.5
41.3
26.8
25.8
outer ϕ = 1.80
coax 40.7
inner ϕ = .48
outer ϕ = 1.65
3.8
feed point
26.3
19.7
5.8
5.8
24.6
14.0
61.1
ϕ = 6.6
6.2
22.7
25.6
"W"
12.8
6
fin info (See attached graph paper with green graph paper)
• outside edge is thicker : 6.1
• most of fin is thin : 3.5
feed at: 10 (V-pol)
11 (H-pol)
Measurements origin is at base, front corner
change on back? Assume front
z front back
0 0 22
6 0 22
6 -2.3 22
9 -2.3 22
9 -8.1 22
(back) 28.3 -8.1 22 (last fixed point)
34 -8.0 ~24
54 -7.0 ~32
68 -5.8 ~37.5
80 -4.1 ~42
99.5 -0.5 ~49.5
110 2.5 ~53.5
130 10.0 ~61
137 12.5 ~63.5
(back) also front 144.8 17.0 66.5
150 20.5 ~66
(back) 156 27 64.5
157 28 63.5
if "~", should calculate
continued
z front back
161 35 60
163 40 53
163.5 50.5 50.5
Top

Figure 3.1.2 a

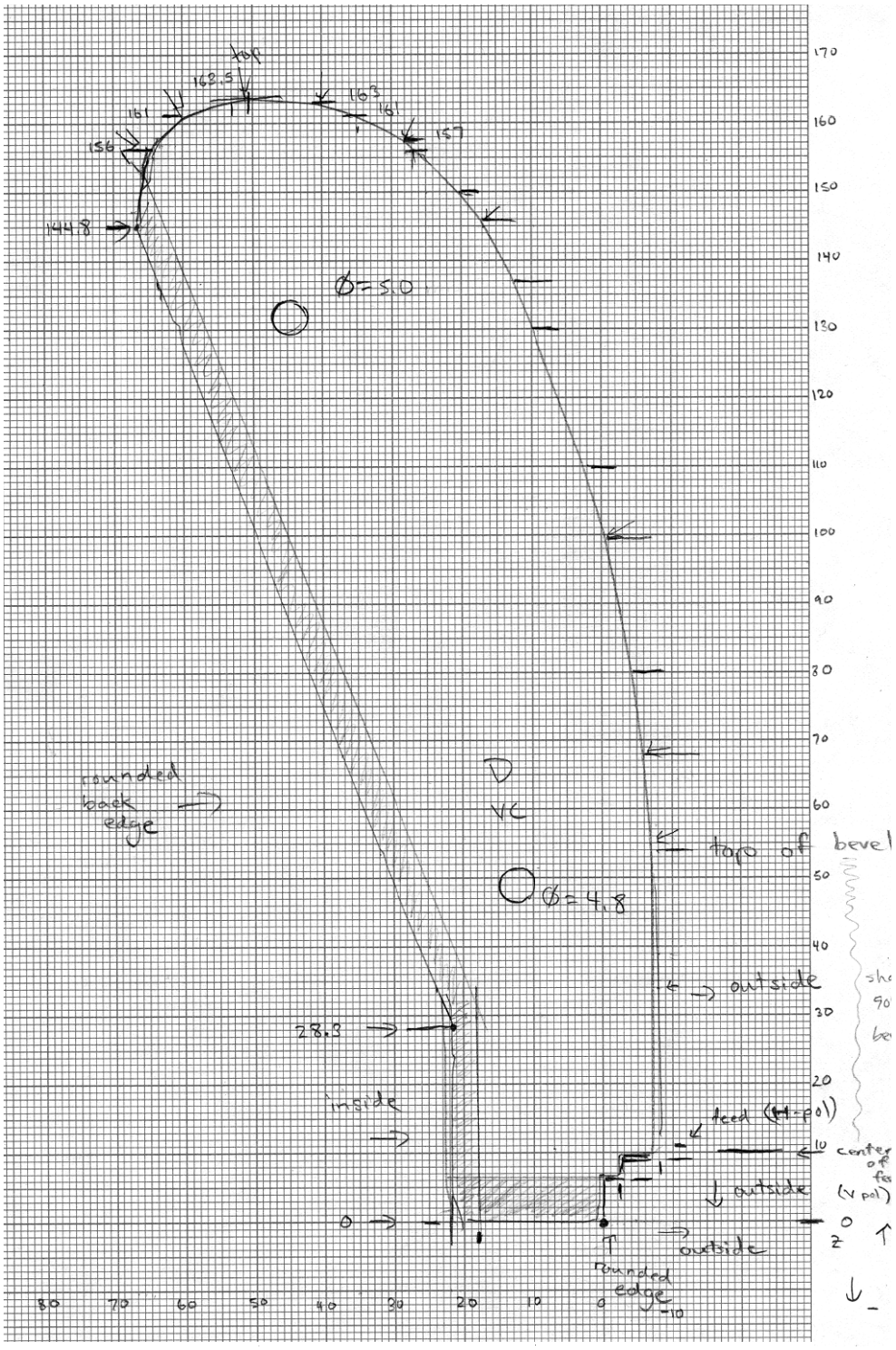

top
168.5
163
161
161
157
156
144.8
Ø=5.0
rounded back edge
D
VC
Ø=4.8
top of bevel
outside
28.3
inside
feed
outside
rounded edge
outside
z
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
-10
80
70
60
50
40
30
20
10
0

Figure 3.1.2 b

Figure 3.1.2 c shows the shape of the beveled edge of the fin inside the slot. In addition, the measurements used to model the beveled edge are shown.

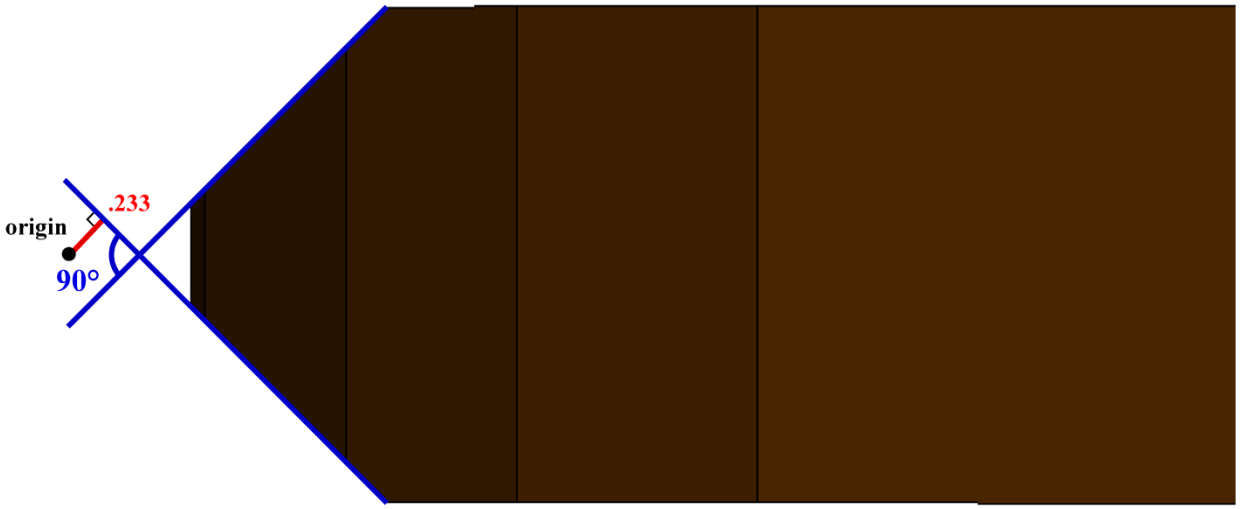


Figure 3.1.2 c

Here is a summary of the measurements. These are the exact variables used to make the CST model.

Table 3.1.2 a

Name	Value (mm)	Description
gap_w	0.686*2	width of the gap between opposing fins
fin_x_shift	gap_w/2	x position to shift the fin coordinate origin
fin_z_shift	-1	z position to shift the fin coordinate origin
fin_thick	3.5	thickness of the fins
base_h	41.3	outer height of the base box, not including the base_top
base_w	69.1	outer width of the base box
base_z	-3+fin_z_shift	z position of the bottom of the inside cutout of the base box
cut_depth	22.7	depth of the inside cutout of the base box
cut_w	53.8	width of the inside cutout of the base box
hemi_r	12.3/2	radius of the hemisphere cutout in the base box
base_top_dia	127.1	diameter of circle that makes up the top of the base
base_top_thick	3.175	thickness of the circle that makes up the top of the base
base_top_cutout	cut_w	side length of the square cutout in the circle that makes up the top of the base

For the model that includes the radiation shield and absorber, here are the values used.

Table 3.1.2 b

Name	Value (mm)	Description
feed_top	z15 (154.5)	z position of the top of the feed by itself
feed_bot	base_z-(base_h-cut_depth)	z position of the bottom of the feed by itself
shld_ir	115-shld_thick	inner radius of the radiation shield
shld_thick	0.864	thickness of the sides of the radiation shield
shld_base_thick	6	thickness of the base plate under the feed
abs_thick	10	thickness of the absorber material

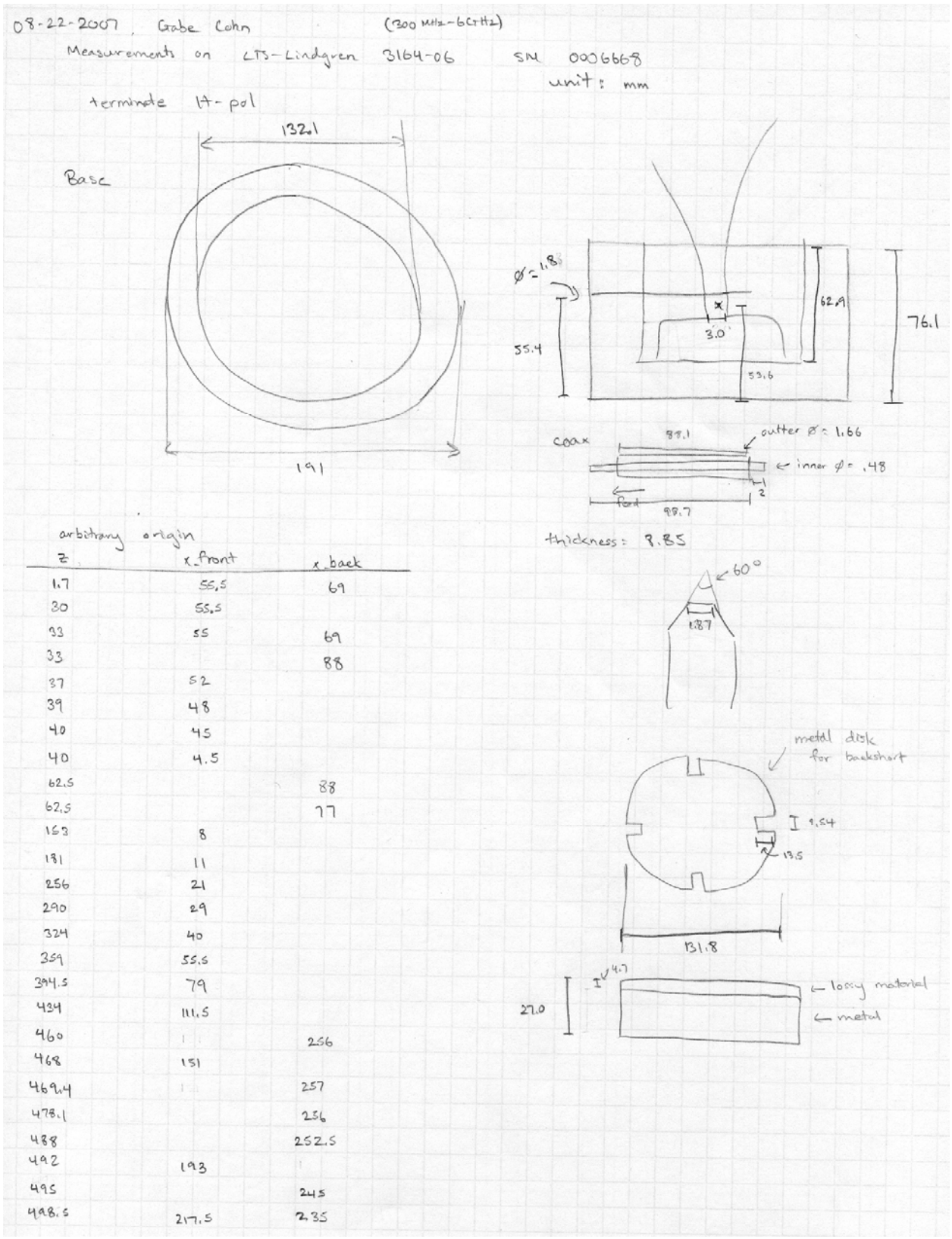
In CST, the fin shape is defined by a set of points that trace out a curve. The following points define this curve. The origin of the coordinate system is the edge of the fin at the bottom of the gap.

Table 3.1.2 c

point number	z position	x position
0	0	0
1	25	0.1
2	45	1.1
3	59	2.3
4	71	4
5	90.5	7.6
6	101	10.6
7	121	18.1
8	128	20.6
9	135.8	25.1
10	141	28.6
11	147	35.1
12	148	36.1
13	152	43.1
14	154	48.1
15	154.5	58.6
16	154	61.1
17	152	68.1
18	148	71.6
19	147	72.6
20	135.8	74.6
21	19.3	30.1
22	-9	30.1
23	-9	8.1
24	-3	8.1
25	-3	5.8
26	0	5.8
27	0	5.8
28	0	0

3.1.3 3164-06 Physical Measurements

Almost all measurements were made with calipers, with the exception of the fin shape. The fin shape was determined my tracing its outline on graph paper, and then making measurements from the image. The following sheets are my original notes from measuring the 3164-06 feed.



arbitrary origin

z	x_front	x_back
1.7	55.5	69
30	55.5	
33	55	69
33		88
37	52	
39	48	
40	45	
40	4.5	
62.5		88
62.5		77
163	8	
181	11	
256	21	
290	29	
324	40	
359	55.5	
394.5	79	
434	111.5	
460		256
468	151	
469.4		257
478.1		256
488		252.5
492	193	
495		245
498.5	217.5	235

Figure 3.1.3 a

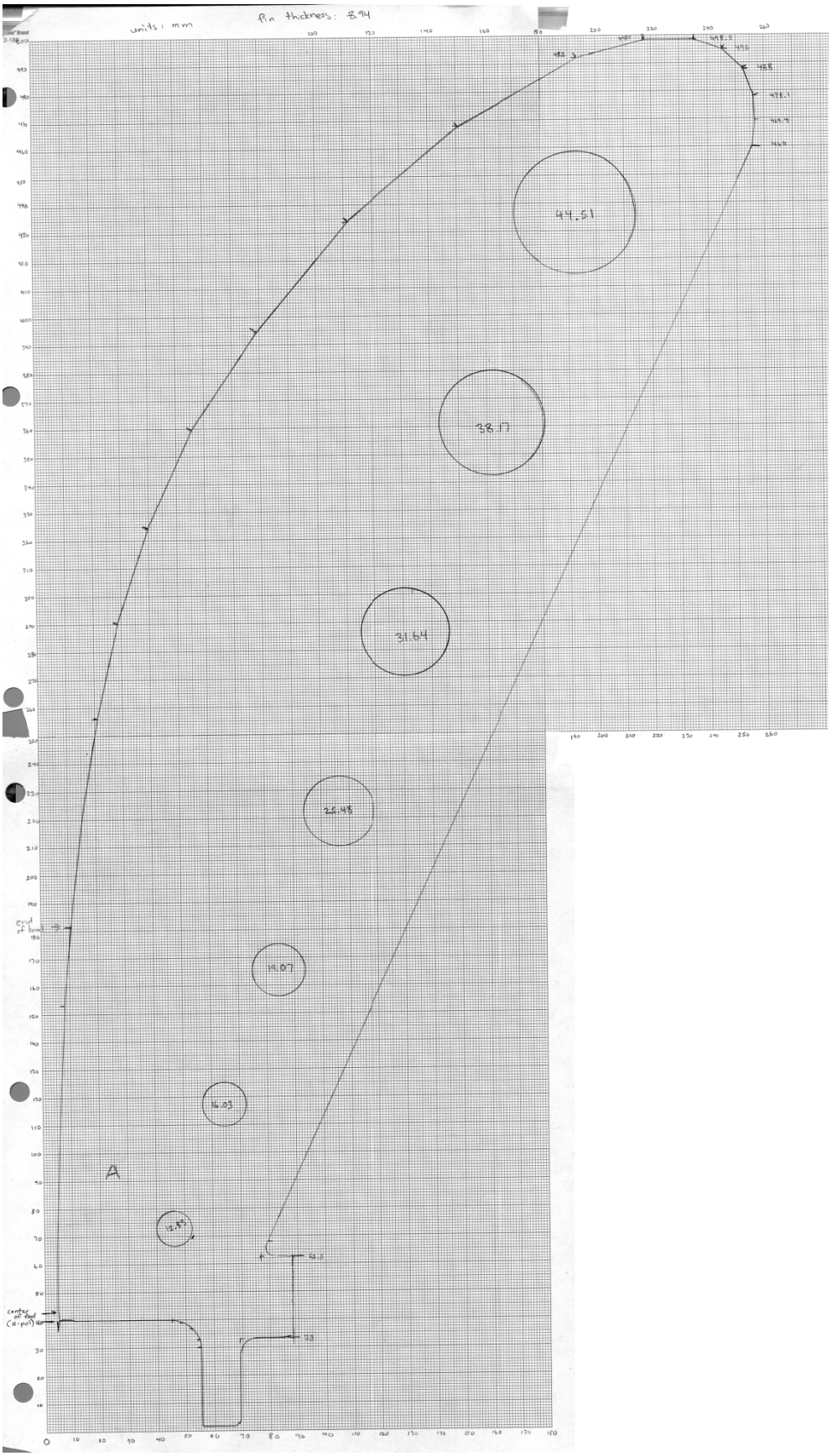

units: mm
Pin thickness: 8.94
44.51
38.17
31.64
25.48
19.07
16.03
12.85
A

Figure 3.1.3 b

Figure 3.1.3 c shows the shape of the beveled edge of the fin inside the slot. In addition, the measurements used to model the beveled edge are shown.

Figure 3.1.3 c

Here is a summary of the measurements. These exact variables are used to make the CST model.

Table 3.1.3 a

Name	Value (mm)	Description
gap_w	3.0	width of the gap between opposing fins
fin_x_shift	gap_w/2	x position to shift the fin coordinate origin
fin_z_shift	-3	z position to shift the fin coordinate origin
fin_thick	8.85	thickness of the fins
base_h	76.1	outer height of the base box, not including the base_top
base_or	191/2	radius of the outside of the base box
base_ir	132.1/2	radius of the inside cutout of the base box
base_z	-38.3+fin_z_shift	z position of the bottom of the inside cutout of the base box
cut_depth	62.9	depth of the inside cutout of the base box
fin_slot_edge	1.87	thickness of fin at the edge of the fin at the slot (tip of bevel)
bevel_shift	(gap_w-(1.732*fin_slot_edge))/4	See description in section 3.4.1. assumes 60 degree bevel angle

For the model that includes the radiation shield and absorber, here are the values used in the CST model.

Table 3.1.3 b

Name	Value (mm)	Description
feed_top	458.5	z position of the top of the feed by itself
feed_bot	base_z-(base_h-cut_depth)	z position of the bottom of the feed by itself
shld_ir	355.5 (14”)	inner radius of the radiation shield
shld_thick	2*max_cellsize	thickness of the sides of the radiation shield
shld_base_thick	2*max_cellsize	thickness of the base plate under the feed
abs_thick	10	thickness of the absorber material

In CST, the fin shape is defined by a set of points that trace out a curve. The following points define this curve. The origin of the coordinate system is the edge of the fin at the bottom of the gap.

Table 3.1.3 c

point number	z position	x position
0	0	0
1	113	3.5
2	141	6.5
3	216	16.5
4	250	24.5
5	284	35.5
6	319	51
7	354.5	74.5
8	394	107
9	428	146.5
10	452	188.5
11	458.5	213
12	458.5	230.5
13	455	240.5
14	448	248
15	438.1	251.5
16	429.4	252.5
17	420	251.5
18	22.5	72.5
19	22.5	83.5
20	-7	83.5
21	-7	64.5
22	-38.3	64.5
23	-38.3	51
24	-10	51
25	-7	50.5
26	-3	47.5
27	-1	43.5
28	0	40.5
29	0	0

3.2 Modeling Microwave Absorber

One of the main challenges of modeling the feed in an absorber lined radiation shield was getting an accurate model of the microwave absorber. Both QuickWave and CST allow materials to be defined using the following material properties: relative permittivity (ε_r), relative permeability

(μ_r), electric conductivity (σ), and magnetic loss (σ_m). However, very few manufactures of microwave absorber report these values on the data sheets. The microwave absorber that I am trying to model is Eccosorb AN-73 made by Emerson & Cuming Microwave Products. Emerson & Cuming does not seem to list this data on their webpage [3-1]. As a result, I created a test structure in QuickWave to test the behavior of materials with different properties. I watched the E-fields as they propagated through the material. I concluded on using the following parameters:

$\varepsilon_r = 5$
$\mu_r = 5$
$\sigma = 1.2$ S/m
$\sigma_m = 1.2$ Ω/m

These parameters appear to give a fairly good model of the AN-73 based on pattern data.

In the future, we are hoping to make an apparatus to test absorber material at cryogenic temperatures and obtain the material properties. This should dramatically increase the accuracy of the simulation results involving microwave-absorbing materials.

3.3 QuickWave Model

3.3.1 How to build the model

QuickWave was only used to model the Inventor file design of the 3164-05 (see section 3.1.1), so this will only refer to that particular design.

Table 3.1.1 c defines the complete shape of a single fin. This table is stored in a Microsoft Excel file, which can be saved in plain text, tab delimited format. This tab delimited file can be parsed to create the QuickWave UDO language code that will create a single fin. The function named run2() in the python script makeFin.py will create a new plain text file which contains the UDO code (see section 3.3.2). This code can be copied into the file 3164-05_fin.udo, which will create a single fin of the structure (see section 3.3.3). This object will be called inside the file Lindgren3164-05.udo, which defines the entire structure (see section 3.3.3).

Open a new QuickWave project. Define the background material and units. Make sure that the program is in draft mode. Now insert the UDO object from the file named Lindgren3164-05.udo. There are several options, which allow you to select which parts of the structure are drawn, which ports to define, the frequency range, mesh size, and far-field frequencies to evaluate. For more information on these options, see section 3.3.3. Once the structure is imported, switch to final mode. This will draw the structure again (this may take several minutes). When this is finished, ensure that the mesh is appropriate. If the mesh needs to be modified, go back to draft mode, make all changes, and then go back to final mode and check again. Once the structure and mesh are fully defined, the structure can be exported for simulation. This process is very straightforward because the UDO code will do all of the work for you.

3.3.2 Scripts to create QuickWave UDO code

The python file makeFin.py contains all functions for creating the QuickWave UDO code to draw the fin shape of the 3164-05 Inventor file design. Each function is described below in detail. The main function is run2().

Function:	run
Arguments:	infile – name of input file outfile – name of output file
Inputs:	tab delimited table of measurements from the fin, not discussed in this document
Outputs:	plain text file containing UDO language code
Description:	Old, not used anymore. Creates cutting blocks that are used to create a fin by making cuts into a rectangular block of metal. The input file is a format that is no longer used and is not discussed in this document. This method of creating fins does not work, due to meshing errors and it is therefore no longer used.

Function:	run2
Arguments:	infile – name of input file outfile – name of output file
Inputs:	tab delimited table of the same form as Table 3.1.1 c
Outputs:	plain text file containing UDO language code
Description:	Creates the UDO code to build a fin using vertical slices. This method is used to create the fin for the 3164-05 Inventor files design.

Function:	run3
Arguments:	infile – name of input file outfile – name of output file
Inputs:	tab delimited table of the same form as Table 3.1.1 c
Outputs:	plain text file containing UDO language code
Description:	Used for experimenting with fin thickness. Creates the UDO code to build a fin using vertical slices. However, the fin will have two different thicknesses, with the transition point at a fixed position. The fin thicknesses and transition point are defined in the python code. Used to create the structure described in section 6.5.3.

Function:	run4
Arguments:	infile – name of input file outfile – name of output file
Inputs:	tab delimited table of the same form as Table 3.1.1 c
Outputs:	plain text file containing UDO language code
Description:	Used for experimenting with fin thickness. Creates the UDO code to build a fin using vertical slices. However, the fin will have two different thicknesses, with a transition point that moves so that it is always a fixed distance from the fin edge. The fin thicknesses and fixed transition distance are defined in the python code. Used to create the fins described in section 6.5.4.

Function:	udo2py
Arguments:	infile – name of input file outfile – name of output file
Inputs:	plain text file containing UDO language code
Outputs:	plain text file containing python code
Description:	Used to create all of the above functions. This will parse a file with UDO language code and produce the python code that will generate the UDO language code in the input file.

The following pages contain the source code of the makeFin.py script.

makeFin.py 9/12/2007

```
# script to automate the making of UDO calls for creating the fin in QuickWave

def run(infile="measurements_tab.txt", outfile="fin_udo_calls.txt"):
    try:
        inf = file(infile, 'r')
    except:
        print infile, "not found."
        return
    try:
        outf = file(outfile, 'w')
    except:
        print "can't create", outfile
        return

    for i in range(3):
        line = inf.readline()   # skip the first 3 lines

    calls = []
    marks = []
    dels = []
    line = inf.readline()
    while line:
        line = line.split()
        if len(line):
            try:
                calls.append("CALL(\"basic/solidvb.udo\", px_fin_cut%s, %s, %s, %s, %s, air, %s, %s, x+(%s), y, z+(%s), 12);\n" %(line[0], line[1], line[2], line[3], line[4], line[6], line[5], line[9], line[11]))
                marks.append("MARKFJ(OBJECT, px_fin_cut%s, ACTIVE);\n" %line[0])
                dels.append("DELETE(OBJECT, px_fin_cut%s);\n" %line[0])
            except:
                print "error"
                print line
                break
        line = inf.readline()   # get next line

    outf.writelines(calls)
    outf.write('\n\n')
    outf.writelines(marks)
    outf.write('\n\n')
    outf.writelines(dels)

    inf.close()
    outf.close()
    return

def run2(infile="sectors_tab.txt", outfile="fin_udo.txt"):
    '''makes fins from slices'''
    try:
        inf = file(infile, 'r')
    except:
        print infile, "not found."
        return
    try:
```

-1-

makeFin.py 9/12/2007

```
53          outf = file(outfile, 'w')
54      except:
55          print "can't create", outfile
56          return
57
58      line = inf.readline()   # skip the first line
59      lines = inf.readlines()
60      inf.close()
61
62      firstline = lines[0].split()
63      i = -1
64      lastline = lines[i].split()
65      while not len(lastline):
66          i -= 1
67          lastline = lines[i].split()
68      angleline = lines[3].split()
69
70      outf.write("# generate fin using consecutive layers of combined elements starting at the bottom\n\n")
71      outf.write("fin_top = %s;\n" %lastline[4])
72      outf.write("fin_bot = %s;\n" %firstline[1])
73      outf.write("fin_width = 3.615;\n")
74      outf.write("fin_w2 = fin_width / 2;\n")
75      outf.write("buff = 0.00002; # add to level values before calling MESHINDEX to be sure to include the desired level\n")
76
77  ##    outf.write("\n# insert mesh spapping planes at critical locations\n")
78  ##    outf.write("CALL(\"elements/specz.udo\", fin_topM, 0, x, y, z+fin_top, 6);\n")
79  ##    outf.write("CALL(\"elements/specz.udo\", fin_botM, 0, x, y, z+fin_bot, 6);\n")
80  ##    outf.write("CALL(\"elements/specz.udo\", fin_excitbaseM, 0, x, y, z+(%s), 6);\n" %firstline[4])
81  ##    outf.write("CALL(\"elements/specz.udo\", fin_backangleM, 0, x, y, z+(%s), 6);\n" %angleline[1])
82
83      outf.write("\nif SLICINGPHASE == 1 do # if in final phase\n")
84      outf.write("\ttop_index = MESHINDEX(z+fin_top+buff);\n")
85      outf.write("\tTEST(top_index > 0, \"Mesh not ready for FINAL phase drawing\");\n")
86      outf.write("\tcur_index = MESHINDEX(z+fin_bot+buff); # start at the bottom of the fin\n")
87      outf.write("\tif cur_index < 0 do\n\t\tcur_index = 0;\n\tendif;\n")
88      outf.write("\tcur_fx = %s;\n" %firstline[2])
89      outf.write("\tcur_bx = %s;\n" %firstline[3])
90      outf.write("endif; # final phase\n")
91
92      for line in lines:
93          line = line.split()
94          if len(line):
95              try:
96                  outf.write("\n# fin sector %s\n\n" %line[0])
97                  outf.write("# params\n")
98                  outf.write("bot_z = %s; # level (z position) of bottom segment\n" %line[1])
99                  outf.write("bot_fx = %s; # x position of front edge of bottom segment\n" %line[2])
```

-2-

makeFin.py 9/12/2007

```
100                 outf.write("bot_bx = %s; # x position of back edge of bottom segment\n"
     %line[3])
101                 outf.write("top_z = %s; # level (z position) of top segment\n" %line[4])
102                 outf.write("top_fx = %s; # x position of front edge of top segment\n" %
    line[5])
103                 outf.write("top_bx = %s; # x position of back edge of top segment\n" %
    line[6])
104                 outf.write("f_angle = %s; # angle (up from x-axis) of the front edge\n"
     %line[7])
105                 outf.write("b_angle = %s; # angle (up from x-axis) of the back
    edge\n\n" %line[8])
106
107                 # if in draft phase
108                 outf.write("if SLICINGPHASE == 0 do # if in draft phase\n")
109                 # bottom element
110                 outf.write("\tELEMENT(z+bot_z, 0, 15, metal, fin_%s_b, INn); # bottom
    element\n" %line[0])
111                 outf.write("\t\tNEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);\n")
112                 outf.write("\t\tADDY(fin_width);\n")
113                 outf.write("\t\tADDLINE(x+bot_fx, y+fin_w2);\n")
114                 outf.write("\t\tCLOSELINE;\n")
115                 outf.write("\tENDELEM;\n")
116                 # cover (top) element
117                 outf.write("\tELEMENT(z+top_z, 0, 16, metal, fin_%s_c, INn); # cover
    (top) element\n" %line[0])
118                 outf.write("\t\tNEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);\n")
119                 outf.write("\t\tADDY(fin_width);\n")
120                 outf.write("\t\tADDLINE(x+top_fx, y+fin_w2);\n")
121                 outf.write("\t\tCLOSELINE;\n")
122                 outf.write("\tENDELEM;\n")
123                 outf.write("endif; # draft phase\n")
124
125                 # if in final phase
126                 outf.write("\nif SLICINGPHASE == 1 do # if in final phase\n")
127                 outf.write("\ttop_index = MESHINDEX(z+top_z+buff);\n")
128                 outf.write("\tTEST(top_index > 0, \"ERROR: mesh not ready on a for a
    segment\"); # this should never happen, because the mesh should already be setup\n")
129                 outf.write("\tOPENOBJECT(fin_%s);\n" %line[0])
130
131                 # loop through all slices in the sector
132                 outf.write("\t\tcount = 1;\n")
133                 outf.write("\t\twhile cur_index < top_index do\n")
134                 outf.write("\t\t\tbot_level = MESH(cur_index);\n")
135                 outf.write("\t\t\ttop_level = MESH(++cur_index);\n")
136                 # get new x values if at the bottom of the sector
137                 outf.write("\t\t\tif bot_level == (z+bot_z) do\n")
138                 outf.write("\t\t\t\tcur_fx = bot_fx;\n")
139                 outf.write("\t\t\t\tcur_bx = bot_bx;\n")
140                 outf.write("\t\t\tendif;\n")
141                 # bottom element
142                 outf.write("\t\t\tELEMENT(bot_level, 0, 15, metal,
    fin_%s@\"_\"@count@b, INn); # bottom element\n" %line[0])
143                 outf.write("\t\t\t\tNEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);\n")
144                 outf.write("\t\t\t\tADDY(fin_width);\n")
```

makeFin.py **9/12/2007**

```
145                     outf.write("\t\t\t\tADDLINE(cur_fx, y+fin_w2);\n")
146                     outf.write("\t\t\t\tCLOSELINE;\n")
147                     outf.write("\t\t\tENDELEM;\n")
148                     # get new x values
149                     outf.write("\t\t\tcur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));\n")
150                     outf.write("\t\t\tcur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));\n")
151                     # top element
152                     outf.write("\t\t\tELEMENT(top_level, 0, 16, metal, ↵
    fin_%s@\"_\"@count@c, INn); # cover (top) element\n" %line[0])
153                     outf.write("\t\t\t\tNEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);\n")
154                     outf.write("\t\t\t\tADDY(fin_width);\n")
155                     outf.write("\t\t\t\tADDLINE(cur_fx, y+fin_w2);\n")
156                     outf.write("\t\t\t\tCLOSELINE;\n")
157                     outf.write("\t\t\tENDELEM;\n")
158                     outf.write("\t\t\tcount++;\n")
159                     outf.write("\t\tendwhile;\n")
160
161                     outf.write("\tCLOSEOBJ; # fin_%s\n" %line[0])
162                     outf.write("endif; # final phase\n")
163                 except:
164                     print "error"
165                     print line
166                     break
167
168         outf.close()
169         return
170
171     def run3(infile="sectors_tab.txt", outfile="fin_udo.txt"):
172         '''makes fins from slices using two different thicknesses with fixed transition ↵
    point'''
173         try:
174             inf = file(infile, 'r')
175         except:
176             print infile, "not found."
177             return
178         try:
179             outf = file(outfile, 'w')
180         except:
181             print "can't create", outfile
182             return
183
184         line = inf.readline()   # skip the first line
185         lines = inf.readlines()
186         inf.close()
187
188         firstline = lines[0].split()
189         i = -1
190         lastline = lines[i].split()
191         while not len(lastline):
192             i -= 1
193             lastline = lines[i].split()
194         angleline = lines[3].split()
195
196         outf.write("# generate fin using consecutive layers of combined elements starting ↵
```

makeFin.py 9/12/2007

```
    at the bottom\n\n")
197     outf.write("fin_top = %s;\n" %lastline[4])
198     outf.write("fin_bot = %s;\n" %firstline[1])
199     outf.write("fin_gap = 0.686; # half of the width of the gap\n")
200     outf.write("fin_width_outer = 3.615;\n")
201     outf.write("fin_w2_outer = fin_width_outer / 2;\n")
202     outf.write("fin_width_inner = 2 * fin_gap;\n")
203     outf.write("fin_w2_inner = fin_width_inner / 2;\n")
204     outf.write("fin_edge_trans = 2.8; # transition to thin fin\n")
205     outf.write("buff = 0.00002; # add to level values before calling MESHINDEX to be
    sure to include the desired level\n")
206
207 ##    outf.write("\n# insert mesh spapping planes at critical locations\n")
208 ##    outf.write("CALL(\"elements/specz.udo\", fin_topM, 0, x, y, z+fin_top, 6);\n")
209 ##    outf.write("CALL(\"elements/specz.udo\", fin_botM, 0, x, y, z+fin_bot, 6);\n")
210 ##    outf.write("CALL(\"elements/specz.udo\", fin_excitbaseM, 0, x, y, z+(%s), 6);\n"
    %firstline[4])
211 ##    outf.write("CALL(\"elements/specz.udo\", fin_backangleM, 0, x, y, z+(%s), 6);\n"
    %angleline[1])
212
213     outf.write("\nif SLICINGPHASE == 1 do # if in final phase\n")
214     outf.write("\ttop_index = MESHINDEX(z+fin_top+buff);\n")
215     outf.write("\tTEST(top_index > 0, \"Mesh not ready for FINAL phase drawing\");\n")
216     outf.write("\tcur_index = MESHINDEX(z+fin_bot+buff); # start at the bottom of the
    fin\n")
217     outf.write("\tif cur_index < 0 do\n\t\tcur_index = 0;\n\tendif;\n")
218     outf.write("\tcur_fx = %s;\n" %firstline[2])
219     outf.write("\tcur_bx = %s;\n" %firstline[3])
220     outf.write("endif; # final phase\n")
221
222     for line in lines:
223         line = line.split()
224         if len(line):
225             try:
226                 outf.write("\n# fin sector %s\n\n" %line[0])
227                 outf.write("# params\n")
228                 outf.write("bot_z = %s; # level (z position) of bottom segment\n" %line
    [1])
229                 outf.write("bot_fx = %s; # x position of front edge of bottom
    segment\n" %line[2])
230                 outf.write("bot_bx = %s; # x position of back edge of bottom segment\n"
     %line[3])
231                 outf.write("top_z = %s; # level (z position) of top segment\n" %line[4])
232                 outf.write("top_fx = %s; # x position of front edge of top segment\n" %
    line[5])
233                 outf.write("top_bx = %s; # x position of back edge of top segment\n" %
    line[6])
234                 outf.write("f_angle = %s; # angle (up from x-axis) of the front edge\n"
     %line[7])
235                 outf.write("b_angle = %s; # angle (up from x-axis) of the back
    edge\n\n" %line[8])
236
237                 # if in draft phase
238                 outf.write("if SLICINGPHASE == 0 do # if in draft phase\n")
```

makeFin.py 9/12/2007

```
239                 # bottom element
240                 outf.write("\tedge_trans = fin_edge_trans;\n")
241                 outf.write("\touter_w2 = fin_w2_outer;\n")
242                 outf.write("\tif fin_edge_trans < bot_fx do\n")
243                 outf.write("\t\tedge_trans = bot_fx;\n")
244                 outf.write("\t\touter_w2 = fin_w2_inner;\n")
245                 outf.write("\tendif;\n")
246                 outf.write("\tELEMENT(z+bot_z, 0, 15, metal, fin_%s_b, INn); # bottom element\n" %line[0])
247                 outf.write("\t\tNEWLINE(x+bot_fx, y-outer_w2, x+edge_trans, y-outer_w2);\n")
248                 outf.write("\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
249                 outf.write("\t\tADDLINE(x+bot_bx, y-fin_w2_inner);\n")
250                 outf.write("\t\tADDY(fin_width_inner);\n")
251                 outf.write("\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
252                 outf.write("\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
253                 outf.write("\t\tADDLINE(x+bot_fx, y+outer_w2);\n")
254                 outf.write("\t\tCLOSELINE;\n")
255                 outf.write("\tENDELEM;\n")
256                 # cover (top) element
257                 outf.write("\tedge_trans = fin_edge_trans;\n")
258                 outf.write("\touter_w2 = fin_w2_outer;\n")
259                 outf.write("\tif fin_edge_trans < top_fx do\n")
260                 outf.write("\t\tedge_trans = top_fx;\n")
261                 outf.write("\t\touter_w2 = fin_w2_inner;\n")
262                 outf.write("\tendif;\n")
263                 outf.write("\tELEMENT(z+top_z, 0, 16, metal, fin_%s_c, INn); # cover (top) element\n" %line[0])
264                 outf.write("\t\tNEWLINE(x+top_fx, y-outer_w2, x+edge_trans, y-outer_w2);\n")
265                 outf.write("\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
266                 outf.write("\t\tADDLINE(x+top_bx, y-fin_w2_inner);\n")
267                 outf.write("\t\tADDY(fin_width_inner);\n")
268                 outf.write("\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
269                 outf.write("\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
270                 outf.write("\t\tADDLINE(x+top_fx, y+outer_w2);\n")
271                 outf.write("\t\tCLOSELINE;\n")
272                 outf.write("\tENDELEM;\n")
273                 outf.write("endif; # draft phase\n")
274
275                 # if in final phase
276                 outf.write("\nif SLICINGPHASE == 1 do # if in final phase\n")
277                 outf.write("\ttop_index = MESHINDEX(z+top_z+buff);\n")
278                 outf.write("\tTEST(top_index > 0, \"ERROR: mesh not ready on a for a segment\"); # this should never happen, because the mesh should already be setup\n")
279                 outf.write("\tOPENOBJECT(fin_%s);\n" %line[0])
280
281                 # loop through all slices in the sector
282                 outf.write("\t\tcount = 1;\n")
283                 outf.write("\t\twhile cur_index < top_index do\n")
284                 outf.write("\t\t\tbot_level = MESH(cur_index);\n")
285                 outf.write("\t\t\ttop_level = MESH(++cur_index);\n")
286                 # get new x values if at the bottom of the sector
287                 outf.write("\t\t\tif bot_level == (z+bot_z) do\n")
```

makeFin.py **9/12/2007**

```
                outf.write("\t\t\t\tcur_fx = bot_fx;\n")
                outf.write("\t\t\t\tcur_bx = bot_bx;\n")
                outf.write("\t\t\tendif;\n")
                # bottom element
                outf.write("\t\t\tedge_trans = fin_edge_trans;\n")
                outf.write("\t\t\touter_w2 = fin_w2_outer;\n")
                outf.write("\t\t\tif fin_edge_trans < cur_fx do\n")
                outf.write("\t\t\t\tedge_trans = cur_fx;\n")
                outf.write("\t\t\t\touter_w2 = fin_w2_inner;\n")
                outf.write("\t\t\tendif;\n")
                outf.write("\t\t\tELEMENT(bot_level, 0, 15, metal, fin_%s@\"_\"@count@b, INn); # bottom element\n" %line[0])
                outf.write("\t\t\t\tNEWLINE(cur_fx, y-outer_w2, x+edge_trans, y-outer_w2);\n")
                outf.write("\t\t\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
                outf.write("\t\t\t\tADDLINE(cur_bx, y-fin_w2_inner);\n")
                outf.write("\t\t\t\tADDY(fin_width_inner);\n")
                outf.write("\t\t\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
                outf.write("\t\t\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
                outf.write("\t\t\t\tADDLINE(cur_fx, y+outer_w2);\n")
                outf.write("\t\t\t\tCLOSELINE;\n")
                outf.write("\t\t\tENDELEM;\n")
                # get new x values
                outf.write("\t\t\tcur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));\n")
                outf.write("\t\t\tcur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));\n")
                # top element
                outf.write("\t\t\tedge_trans = fin_edge_trans;\n")
                outf.write("\t\t\touter_w2 = fin_w2_outer;\n")
                outf.write("\t\t\tif fin_edge_trans < cur_fx do\n")
                outf.write("\t\t\t\tedge_trans = cur_fx;\n")
                outf.write("\t\t\t\touter_w2 = fin_w2_inner;\n")
                outf.write("\t\t\tendif;\n")
                outf.write("\t\t\tELEMENT(top_level, 0, 16, metal, fin_%s@\"_\"@count@c, INn); # cover (top) element\n" %line[0])
                outf.write("\t\t\t\tNEWLINE(cur_fx, y-outer_w2, x+edge_trans, y-outer_w2);\n")
                outf.write("\t\t\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
                outf.write("\t\t\t\tADDLINE(cur_bx, y-fin_w2_inner);\n")
                outf.write("\t\t\t\tADDY(fin_width_inner);\n")
                outf.write("\t\t\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
                outf.write("\t\t\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
                outf.write("\t\t\t\tADDLINE(cur_fx, y+outer_w2);\n")
                outf.write("\t\t\t\tCLOSELINE;\n")
                outf.write("\t\t\tENDELEM;\n")
                outf.write("\t\t\tcount++;\n")
                outf.write("\t\tendwhile;\n")

                outf.write("\tCLOSEOBJ; # fin_%s\n" %line[0])
                outf.write("endif; # final phase\n")
            except:
                print "error"
                print line
                break

```

makeFin.py 9/12/2007

```
338         outf.close()
339         return
340
341     def run4(infile="sectors_tab.txt", outfile="fin_udo.txt"):
342         '''makes fins from slices using two different thicknesses, thick along whole slot edge'''
343         try:
344             inf = file(infile, 'r')
345         except:
346             print infile, "not found."
347             return
348         try:
349             outf = file(outfile, 'w')
350         except:
351             print "can't create", outfile
352             return
353
354         line = inf.readline()   # skip the first line
355         lines = inf.readlines()
356         inf.close()
357
358         firstline = lines[0].split()
359         i = -1
360         lastline = lines[i].split()
361         while not len(lastline):
362             i -= 1
363             lastline = lines[i].split()
364         angleline = lines[3].split()
365
366         outf.write("# generate fin using consecutive layers of combined elements starting at the bottom\n\n")
367         outf.write("fin_top = %s;\n" %lastline[4])
368         outf.write("fin_bot = %s;\n" %firstline[1])
369         outf.write("fin_gap = 0.686; # half of the width of the gap\n")
370         outf.write("fin_width_outer = 3.615;\n")
371         outf.write("fin_w2_outer = fin_width_outer / 2;\n")
372         outf.write("fin_width_inner = 2 * fin_gap;\n")
373         outf.write("fin_w2_inner = fin_width_inner / 2;\n")
374         outf.write("fin_bevel_edge = 2.030;\n")
375         outf.write("fin_edge_len = 1; # length of thick edge on the slot edge of fin\n")
376         outf.write("buff = 0.00002; # add to level values before calling MESHINDEX to be sure to include the desired level\n")
377
378     ##    outf.write("\n# insert mesh spapping planes at critical locations\n")
379     ##    outf.write("CALL(\"elements/specz.udo\", fin_topM, 0, x, y, z+fin_top, 6);\n")
380     ##    outf.write("CALL(\"elements/specz.udo\", fin_botM, 0, x, y, z+fin_bot, 6);\n")
381     ##    outf.write("CALL(\"elements/specz.udo\", fin_excitbaseM, 0, x, y, z+(%s), 6);\n" %firstline[4])
382     ##    outf.write("CALL(\"elements/specz.udo\", fin_backangleM, 0, x, y, z+(%s), 6);\n" %angleline[1])
383
384         outf.write("\nif SLICINGPHASE == 1 do # if in final phase\n")
385         outf.write("\ttop_index = MESHINDEX(z+fin_top+buff);\n")
386         outf.write("\tTEST(top_index > 0, \"Mesh not ready for FINAL phase drawing\");\n")
```

-8-

makeFin.py 9/12/2007

```
387         outf.write("\tcur_index = MESHINDEX(z+fin_bot+buff); # start at the bottom of the fin\n")
388         outf.write("\tif cur_index < 0 do\n\t\tcur_index = 0;\n\tendif;\n")
389         outf.write("\tcur_fx = %s;\n" %firstline[2])
390         outf.write("\tcur_bx = %s;\n" %firstline[3])
391         outf.write("endif; # final phase\n")
392 
393         for line in lines:
394             line = line.split()
395             if len(line):
396                 try:
397                     outf.write("\n# fin sector %s\n\n" %line[0])
398                     outf.write("# params\n")
399                     outf.write("bot_z = %s; # level (z position) of bottom segment\n" %line[1])
400                     outf.write("bot_fx = %s; # x position of front edge of bottom segment\n" %line[2])
401                     outf.write("bot_bx = %s; # x position of back edge of bottom segment\n" %line[3])
402                     outf.write("top_z = %s; # level (z position) of top segment\n" %line[4])
403                     outf.write("top_fx = %s; # x position of front edge of top segment\n" %line[5])
404                     outf.write("top_bx = %s; # x position of back edge of top segment\n" %line[6])
405                     outf.write("f_angle = %s; # angle (up from x-axis) of the front edge\n" %line[7])
406                     outf.write("b_angle = %s; # angle (up from x-axis) of the back edge\n\n" %line[8])
407 
408                     # if in draft phase
409                     outf.write("if SLICINGPHASE == 0 do # if in draft phase\n")
410                     # bottom element
411                     outf.write("\tedge_trans = bot_fx+fin_edge_len;\n")
412                     outf.write("\touter_w2 = fin_w2_outer;\n")
413                     outf.write("\tif edge_trans < fin_bevel_edge+fin_edge_len do\n")
414                     outf.write("\t\tedge_trans = fin_bevel_edge+fin_edge_len;\n")
415                     outf.write("\tendif;\n")
416                     outf.write("\tif edge_trans > bot_bx do\n")
417                     outf.write("\t\tedge_trans = bot_bx;\n")
418                     outf.write("\tendif;\n")
419                     outf.write("\tELEMENT(z+bot_z, 0, 15, metal, fin_%s_b, INn); # bottom element\n" %line[0])
420                     outf.write("\t\tNEWLINE(x+bot_fx, y-outer_w2, x+edge_trans, y-outer_w2);\n")
421                     outf.write("\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
422                     outf.write("\t\tADDLINE(x+bot_bx, y-fin_w2_inner);\n")
423                     outf.write("\t\tADDY(fin_width_inner);\n")
424                     outf.write("\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
425                     outf.write("\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
426                     outf.write("\t\tADDLINE(x+bot_fx, y+outer_w2);\n")
427                     outf.write("\t\tCLOSELINE;\n")
428                     outf.write("\tENDELEM;\n")
429                     # cover (top) element
430                     outf.write("\tedge_trans = top_fx+fin_edge_len;\n")
```

-9-

makeFin.py 9/12/2007

```
431                 outf.write("\touter_w2 = fin_w2_outer;\n")
432                 outf.write("\tif edge_trans < fin_bevel_edge+fin_edge_len do\n")
433                 outf.write("\t\tedge_trans = fin_bevel_edge+fin_edge_len;\n")
434                 outf.write("\tendif;\n")
435                 outf.write("\tif edge_trans > top_bx do\n")
436                 outf.write("\t\tedge_trans = top_bx;\n")
437                 outf.write("\tendif;\n")
438                 outf.write("\tELEMENT(z+top_z, 0, 16, metal, fin_%s_c, INn); # cover
    (top) element\n" %line[0])
439                 outf.write("\t\tNEWLINE(x+top_fx, y-outer_w2, x+edge_trans,
    y-outer_w2);\n")
440                 outf.write("\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
441                 outf.write("\t\tADDLINE(x+top_bx, y-fin_w2_inner);\n")
442                 outf.write("\t\tADDY(fin_width_inner);\n")
443                 outf.write("\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
444                 outf.write("\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
445                 outf.write("\t\tADDLINE(x+top_fx, y+outer_w2);\n")
446                 outf.write("\t\tCLOSELINE;\n")
447                 outf.write("\tENDELEM;\n")
448                 outf.write("endif; # draft phase\n")
449
450                 # if in final phase
451                 outf.write("\nif SLICINGPHASE == 1 do # if in final phase\n")
452                 outf.write("\ttop_index = MESHINDEX(z+top_z+buff);\n")
453                 outf.write("\tTEST(top_index > 0, \"ERROR: mesh not ready on a for a
    segment\"); # this should never happen, because the mesh should already be setup\n")
454                 outf.write("\tOPENOBJECT(fin_%s);\n" %line[0])
455
456                 # loop through all slices in the sector
457                 outf.write("\t\tcount = 1;\n")
458                 outf.write("\t\twhile cur_index < top_index do\n")
459                 outf.write("\t\t\tbot_level = MESH(cur_index);\n")
460                 outf.write("\t\t\ttop_level = MESH(++cur_index);\n")
461                 # get new x values if at the bottom of the sector
462                 outf.write("\t\t\tif bot_level == (z+bot_z) do\n")
463                 outf.write("\t\t\t\tcur_fx = bot_fx;\n")
464                 outf.write("\t\t\t\tcur_bx = bot_bx;\n")
465                 outf.write("\t\t\tendif;\n")
466                 # bottom element
467                 outf.write("\t\t\tedge_trans = cur_fx+fin_edge_len;\n")
468                 outf.write("\t\t\touter_w2 = fin_w2_outer;\n")
469                 outf.write("\t\t\tif edge_trans < fin_bevel_edge+fin_edge_len do\n")
470                 outf.write("\t\t\t\tedge_trans = fin_bevel_edge+fin_edge_len;\n")
471                 outf.write("\t\t\tendif;\n")
472                 outf.write("\t\t\tif edge_trans > cur_bx do\n")
473                 outf.write("\t\t\t\tedge_trans = cur_bx;\n")
474                 outf.write("\t\t\tendif;\n")
475                 outf.write("\t\t\tELEMENT(bot_level, 0, 15, metal,
    fin_%s@\"_\"@count@b, INn); # bottom element\n" %line[0])
476                 outf.write("\t\t\t\tNEWLINE(cur_fx, y-outer_w2, x+edge_trans,
    y-outer_w2);\n")
477                 outf.write("\t\t\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
478                 outf.write("\t\t\t\tADDLINE(cur_bx, y-fin_w2_inner);\n")
479                 outf.write("\t\t\t\tADDY(fin_width_inner);\n")
```

-10-

makeFin.py 9/12/2007

```
480                     outf.write("\t\t\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
481                     outf.write("\t\t\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
482                     outf.write("\t\t\t\tADDLINE(cur_fx, y+outer_w2);\n")
483                     outf.write("\t\t\t\tCLOSELINE;\n")
484                     outf.write("\t\t\tENDELEM;\n")
485                     # get new x values
486                     outf.write("\t\t\tcur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));\n")
487                     outf.write("\t\t\tcur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));\n")
488                     # top element
489                     outf.write("\t\t\tedge_trans = cur_fx+fin_edge_len;\n")
490                     outf.write("\t\t\touter_w2 = fin_w2_outer;\n")
491                     outf.write("\t\t\tif edge_trans < fin_bevel_edge+fin_edge_len do\n")
492                     outf.write("\t\t\t\tedge_trans = fin_bevel_edge+fin_edge_len;\n")
493                     outf.write("\t\t\tendif;\n")
494                     outf.write("\t\t\tif edge_trans > cur_bx do\n")
495                     outf.write("\t\t\t\tedge_trans = cur_bx;\n")
496                     outf.write("\t\t\tendif;\n")
497                     outf.write("\t\t\tELEMENT(top_level, 0, 16, metal, ↵
    fin_%s@\"_\"@count@c, INn); # cover (top) element\n" %line[0])
498                     outf.write("\t\t\t\tNEWLINE(cur_fx, y-outer_w2, x+edge_trans, ↵
    y-outer_w2);\n")
499                     outf.write("\t\t\t\tADDLINE(x+edge_trans, y-fin_w2_inner);\n")
500                     outf.write("\t\t\t\tADDLINE(cur_bx, y-fin_w2_inner);\n")
501                     outf.write("\t\t\t\tADDY(fin_width_inner);\n")
502                     outf.write("\t\t\t\tADDLINE(x+edge_trans, y+fin_w2_inner);\n")
503                     outf.write("\t\t\t\tADDLINE(x+edge_trans, y+outer_w2);\n")
504                     outf.write("\t\t\t\tADDLINE(cur_fx, y+outer_w2);\n")
505                     outf.write("\t\t\t\tCLOSELINE;\n")
506                     outf.write("\t\t\tENDELEM;\n")
507                     outf.write("\t\t\tcount++;\n")
508                     outf.write("\t\tendwhile;\n")
509
510                     outf.write("\tCLOSEOBJ; # fin_%s\n" %line[0])
511                     outf.write("endif; # final phase\n")
512                 except:
513                     print "error"
514                     print line
515                     break
516
517         outf.close()
518         return
519
520     def udo2py(infile="udo_code.txt", outfile="py_code.txt"):
521         '''takes a file of UDO code, and write the python code to produce the UDO code'''
522         try:
523             inf = file(infile, 'r')
524         except:
525             print infile, "not found."
526             return
527         try:
528             outf = file(outfile, 'w')
529         except:
530             print "can't create", outfile
531             return
```

-11-

makeFin.py 9/12/2007

```
532
533         lines = inf.readlines()
534         inf.close()
535
536         for line in lines:
537             line = line.rstrip() # remove end whitespace
538             line = line.replace("\"", "\\\"") # replace quotes with the quote escape
    character
539             line = line.replace("\t", "\\t") # replace tabs with the tab escape character
540             line = line.replace("    ", "\\t") # replace 4-spaces with the tab escape
    character
541             outf.write("outf.write(\"" + line + "\\n\"")
542             if line.find('%') != -1:
543                 outf.write(" %VAL")
544             outf.write(")\n")
545
546         outf.close()
547         return
548
```

3.3.3 QuickWave UDO Code

Three QuickWave UDO files were created in order to make the model of the Lindgren feed. Each file is described briefly, and then the full source code for each file is presented.

File: **Lindgren3164-05.udo**

Parameters: "object name" – name of object containing entire feed structure
- Type: string
- Default: "antenna"

"Dual Polarized? (1/0)" – draw 2 or 4 fins
- Type: 1 or 0
- Default: 1

"Make Base? (1/0)" – draw the base and fins, or just fins
- Type: 1 or 0
- Default: 1

"Make Cylinders? (1/0)" – draw radiation shield around feed or not
- Type: 1 or 0
- Default: 0

"Make Absorber? (1/0)" – draw absorber lining radiation shield or not
- Type: 1 or 0
- Default: 0

"file describing main port params" – name of excitation port file which describes the main excitation
- Type: string
- Default: "NO" – no port defined…need to define manually

"file describing cross port params" – name of excitation port file which describes the cross excitation (opposite polarization)
- Type: string
- Default: "NO" – no port defined…need to define manually

"Min. freq. (GHz)" – minimum frequency to excite and analyze (GHz)
- Type: floating point
- Default: 1

"Max. freq. (GHz)" – maximum frequency to excite and analyze (GHz)
- Type: floating point
- Default: 18

"Freq. Step (GHz) [-1=AUTO]" – frequency step size for S-parameter analysis (GHz). If -1, then use 15 steps over whole frequency range
- Type: floating point
- Default: 0.02

"Cells per wavelength" – number of mesh cells per wavelength at the highest frequency. This defines the maximum cell size.
- Type: integer
- Default: 15

"NTF Freqs. (GHz)" – space delimited list of frequencies to run far-field patterns
- Type: string: space delimited list of floating point values
- Default: "2 4 6 8"

Description: Draws the entire feed structure and radiation shield with absorber of the Lindgren 3164-05 based on the Inventor files design, depending on options selected. In addition to drawing the structure, this will set-up the mesh, define all needed materials, and set up all post processing. This is a bi-phased (BP) object that should be run in draft mode before switching to final mode.

File: **3164-05_fin.udo**

Parameters: "object name" – name of the fin object
- Type: string
- Default: "fin"

"Rotation" – rotation of fin counter-clockwise around z-axis from the x-axis in degrees
- Type: floating-point
- Default: 0

Description: Draws a single fin of the Lindgren 3164-05 based on the Inventor files design. This is a bi-phased (BP) object that should be run in draft mode before switching to final mode.

File: **sphz.udo**

Parameters: "name" – name of object as seen in the project
- Type: string
- Default: "sp"

"diameter" – diameter of the sphere
- Type: floating-point
- Default: 6

"sectors" – Number of sectors into which length of each horizontal slice of the object is divided. Object's horizontal slice is a curve, which is approximated with n straight lines. The flat section at the bottom and top of the object is introduced because each element (whether combined or regular) must have non-zero width. That is why an arbitrary diameter Δ is used for elements whose diameters would normally be zero (e.g. cover of the top element of the object). The value of Δ is chosen to be equal to the double cell height at the top or bottom of the object. Adjusting the mesh parameters in z-direction changes the Δ value, as well.
- Type: integer
- Default: 16

"medium" – name of the material the object is made of
- Type: string
- Default: air

"mesh snapping planes?" – specifies whether mesh-snapping planes should be added directly above/below the object.
- Type: "Y" or "N"
- Default: "Y"

"2 - combined; <0,1> - regular" – specifies whether each object's layer should be drawn as a combined element (inclined walls indicated by 2) or a regular element (staircase look with value in [0,1]). The parameter value between 0 and 1 makes it possible to define the regular element dimensions in relation to the true taper dimensions measured at the mesh-level directly above and below the given element.

Type: floating-point in [0,1] or 2

Default: 2

Description: Half sphere sliced with combined (BP) or regular elements. This is identical to the built-in object sp.udo, except it is only a half sphere.

Dependencies: Lindgren3164-05.udo requires both 3164-05_fin.udo and sphz.udo.

The source code for all of these UDO language objects is included on the following pages.

Lindgren3164-05.udo – page 60

3154-05_fin.udo – page 65

sphz.udo – page 93

Lindgren3164-05.udo

Lindgren3164-05.udo 9/12/2007

```
1   comment="creates a Lindgren 3164-05 antenna using hardware dimensions from the
    Inventor files";
2   bitmap="nobitmap.bmp";
3   # Gabe Cohn, 07-18-2007
4
5   PAR("object name",oname,"antenna");
6   PAR("Dual Polarized? (1/0)",isDual,1);
7   PAR("Make Base? (1/0)",isBase,1);
8   PAR("Make Cylinders? (1/0)",isCyl,0);
9   PAR("Make Absorder? (1/0)",isAbs,0);
10  PAR("file describing main port params",portf_y,"NO");
11  PAR("file describing cross port params",portf_x,"NO");
12  PAR("Min. freq. (GHz)",fmin,1);
13  PAR("Max. freq. (GHz)",fmax,18);
14  PAR("Freq. Step (GHz) [-1=AUTO]",fstep,0.02);
15  PAR("Cells per wavelength",cpw,15);
16  PAR("NTF Freqs. (GHz)",ntf_freqs,"2 4 6 8");
17
18  ENDHEADER;
19
20  pi = 3.14159265;
21  cellsize = 300/fmax/cpw;
22
23  # fin dimensions
24  fin_top = 152.451;
25  fin_bot = -10.472;
26  fin_width = 3.615;
27  fin_w2 = fin_width / 2;
28  fin_gap = 0.686; # half of the width of the gap
29  fin_max_r = 74.219;
30  base_thick = 3.175;
31  base_bot = fin_bot-base_thick;
32
33  # cylinder and absorber dimensions
34  n_sectors = 32;
35  # radiation shield
36  #shld_thick = .864;
37  shld_thick = 2*cellsize; # use 2*cellsize to be sure that it is meshed correctly
38  shld_outer_r = 115;
39  shld_inner_r = shld_outer_r - shld_thick;
40  shld_inner_bot = base_bot;
41  shld_outer_bot = shld_inner_bot - (6);
42  shld_outer_h = fin_top - shld_outer_bot;
43  shld_inner_h = fin_top - shld_inner_bot;
44  # microwave absorber
45  abs_thick = 10;
46  abs_inner_r = shld_inner_r - abs_thick;
47  abs_inner_h = shld_inner_h - abs_thick;
48  abs_inner_bot = shld_inner_bot + abs_thick;
49
50  # structure dimension limits
51  max_xy = fin_max_r;
52  max_z = fin_top;
53  min_z = fin_bot;  # if no base
```

-1-

Lindgren3164-05.udo 9/12/2007

```
54  if isBase == 1 do
55      min_z = base_bot;  # if base is present
56  endif;
57  if isCyl == 1 do
58      max_xy = shld_outer_r;
59      min_z = shld_outer_bot;
60  endif;
61
62  # absorber medium
63  INSERTMEDIUM("abs_AN-73", DIELECTRIC);
64  eps = 5; # these paramters were found by a QuickWave simulation to be a fairly good
    absorber
65  mu = 5;
66  sigma = 1.2;
67  sigma_m = sigma;
68  MEDIUMPAR("abs_AN-73",eps,mu,sigma,sigma_m,eps,mu,sigma,sigma_m,eps,mu,sigma,sigma_m,0.1
    3);
69  MEDIUMCOL("abs_AN-73",51,51,51,0,1,51,51,51,255,255,255,5);
70
71  # insert mesh spapping planes at critical locations
72
73  # vertical discontinuities
74  CALL("elements/specz.udo", fin_botM, 0.1, x, y, z+fin_bot, 6);
75  CALL("elements/specz.udo", fin_excitbaseM, 0.1, x, y, z+(-0.228), 6);
76  CALL("elements/specz.udo", fin_backangleM, 0.1, x, y, z+(20.079), 6);
77  CALL("elements/specz.udo", fin_topM, 0.1, x, y, z+fin_top, 6);
78
79  # horizontal discontinuities
80  CALL("elements/specy.udo", fin_py_gapM, 0.1, x, y+fin_gap, z, 6);
81  CALL("elements/specy.udo", fin_py_backshortM, 0.1, x, y+(3.510), z, 6);
82  CALL("elements/specy.udo", fin_py_boxedgeM, 0.1, x, y+(26.949), z, 6);
83  CALL("elements/specy.udo", fin_py_topM, 0.1, x, y+(45.617), z, 6);
84  CALL("elements/specy.udo", fin_py_backedgeM, 0.1, x, y+fin_max_r, z, 6);
85
86  CALL("elements/specy.udo", fin_ny_gapM, 0.1, x, y-fin_gap, z, 6);
87  CALL("elements/specy.udo", fin_ny_backshortM, 0.1, x, y+(-3.510), z, 6);
88  CALL("elements/specy.udo", fin_ny_boxedgeM, 0.1, x, y+(-26.949), z, 6);
89  CALL("elements/specy.udo", fin_ny_topM, 0.1, x, y+(-45.617), z, 6);
90  CALL("elements/specy.udo", fin_ny_backedgeM, 0.1, x, y-fin_max_r, z, 6);
91
92  # fin edges and excitation
93  CALL("elements/specy.udo", fin_y_excitationM_y, 0.1, x, y, z, 6);
94  CALL("elements/specx.udo", fin_y_excitationM_o, 0.1, x, y, z, 6);
95  CALL("elements/specx.udo", fin_y_excitationM_px, 0.1, x+fin_gap, y, z, 6);
96  CALL("elements/specx.udo", fin_y_excitationM_nx, 0.1, x-fin_gap, y, z, 6);
97  CALL("elements/specx.udo", fin_y_edgeM_px, 0.1, x+fin_w2, y, z, 6);
98  CALL("elements/specx.udo", fin_y_edgeM_nx, 0.1, x-fin_w2, y, z, 6);
99
100 if isDual == 1 do # add grid lines for other polarization
101     # horizontal discontinuities
102     CALL("elements/specx.udo", fin_px_gapM, 0.1, x+fin_gap, y, z, 6);
103     CALL("elements/specx.udo", fin_px_backshortM, 0.1, x+(3.510), y, z, 6);
104     CALL("elements/specx.udo", fin_px_boxedgeM, 0.1, x+(26.949), y, z, 6);
105     CALL("elements/specx.udo", fin_px_topM, 0.1, x+(45.617), y, z, 6);
```

Lindgren3164-05.udo 9/12/2007

```
106       CALL("elements/specx.udo", fin_px_backedgeM, 0.1, x+fin_max_r, y, z, 6);
107
108       CALL("elements/specx.udo", fin_nx_gapM, 0.1, x-fin_gap, y, z, 6);
109       CALL("elements/specx.udo", fin_nx_backshortM, 0.1, x+(-3.510), y, z, 6);
110       CALL("elements/specx.udo", fin_nx_boxedgeM, 0.1, x+(-26.949), y, z, 6);
111       CALL("elements/specx.udo", fin_nx_topM, 0.1, x+(-45.617), y, z, 6);
112       CALL("elements/specx.udo", fin_nx_backedgeM, 0.1, x-fin_max_r, y, z, 6);
113
114       # fin edges and excitation
115       CALL("elements/specx.udo", fin_x_excitationM_x, 0.1, x, y, z, 6);
116       CALL("elements/specy.udo", fin_x_excitationM_o, 0.1, x, y, z, 6);
117       CALL("elements/specy.udo", fin_x_excitationM_py, 0.1, x, y+fin_gap, z, 6);
118       CALL("elements/specy.udo", fin_x_excitationM_ny, 0.1, x, y-fin_gap, z, 6);
119       CALL("elements/specy.udo", fin_x_edgeM_py, 0.1, x, y+fin_w2, z, 6);
120       CALL("elements/specy.udo", fin_x_edgeM_ny, 0.1, x, y-fin_w2, z, 6);
121   endif; # mesh guides for dual polarization
122
123   # setup mesh
124   MESHPAR(cellsize, cellsize, cellsize, 0, 0, 0, 0, 0, 0, 1);
125
126
127   OPENOBJECT(oname);
128
129   if isCyl == 1 do # draw the cylinders (and absorber, maybe)
130       CALL("basic/cyvo.udo", rad_shld, shld_outer_r, shld_outer_h, n_sectors, metal, N, ↵
      x, y, z+shld_outer_bot, 10);
131       if isAbs == 1 do # draw absorber inside cyclinder
132           CALL("basic/cyvo.udo", shld_abs, shld_inner_r, shld_inner_h, n_sectors, ↵
      "abs_AN-73", N, x, y, z+shld_inner_bot, 10);
133           CALL("basic/cyvo.udo", shld_abs_air, abs_inner_r, abs_inner_h, n_sectors, air, ↵
      N, x, y, z+abs_inner_bot, 10);
134       endif;
135       if isAbs == 0 do
136           CALL("basic/cyvo.udo", shld_air, shld_inner_r, shld_inner_h, n_sectors, air, ↵
      N, x, y, z+shld_inner_bot, 10 );
137       endif;
138   endif;
139
140   if isBase == 1 do   # draw the base
141       # make base_box
142       OPENOBJECT(base);
143
144       # make large box of base_box
145       ELEMENT(z+(-13.647), 33.726, 0, metal, base_box, INe);
146           NEWLINE(x+(-30.1245),y+(-30.1245),x+(30.1245),y+(-30.1245));
147           ADDLINE(x+(30.1245),y+(30.1245));
148           ADDLINE(x+(-30.1245),y+(30.1245));
149           CLOSELINE();
150       ENDELEM;
151
152       # make small box (cut-out) of base_box
153       ELEMENT(z+(-2.502), 22.581, 0, air, base_box_cut0, INe);
154           NEWLINE(x+(-26.9445),y+(-26.9445),x+(26.9445),y+(-26.9445));
155           ADDLINE(x+(26.9445),y+(26.9445));
```

Lindgren3164-05.udo 9/12/2007

```
156           ADDLINE(x+(-26.9445),y+(26.9445));
157           CLOSELINE();
158       ENDELEM;
159
160       # make hemisphere cut-out in base_box
161       dia = 13.02;
162       sectors = dia*pi/cellsize;
163       if sectors < 3 do sectors = 3 endif;
164       CALL("user/sphz.udc", base_box_cut1, dia, sectors, air, Y, 2, x, y, z+(-2.502), 10);
165
166       # create the base_lid
167       radius = 50.8;
168       sectors = radius*2*pi/cellsize;
169       if sectors < 3 do sectors = 3 endif;
170       CALL("basic/cyvob.udo", base_lid, radius, 3.175, sectors, metal, E, x, y, ↵
      z+(20.079), 10);
171
172       # create cut-out for base lid
173       ELEMENT(z+(20.079), 3.175, 0, air, base_lid_cut1, IN);
174           NEWLINE(x+(-30.1245),y+(-30.1245),x+(30.1245),y+(-30.1245));
175           ADDLINE(x+(30.1245),y+(30.1245));
176           ADDLINE(x+(-30.1245),y+(30.1245));
177           CLOSELINE();
178       ENDELEM;
179
180       # cut the square out of base_lid
181       MARKFJ(OBJECT, base_lid, PASSIVE);
182       MARKFJ(ELEM, base_lid_cut1, ACTIVE);
183       JOIN(CUT);
184       DELETE(ELEM, base_lid_cut1);
185       MARKFJ(ELEM, ALL, RESET);
186
187       # make all cuts in base_box
188       MARKFJ(ELEM, base_box, PASSIVE);
189       MARKFJ(ELEM, base_box_cut0, ACTIVE);
190       MARKFJ(OBJECT, base_box_cut1, ACTIVE);
191       JOIN(CUT);
192       DELETE(ELEM, base_box_cut0);
193       DELETE(OBJECT, base_box_cut1);
194       MARKFJ(ELEM, ALL, RESET);
195
196       CLOSEOBJ; # base
197   endif;
198
199   # create the fins
200   CALL("3164-05_fin.udo", py_fin, 90, 0, 0, 0, 6); # create the positive-y fin (py_fin)
201   CALL("3164-05_fin.udo", ny_fin, 270, 0, 0, 0, 6); # create the negative-y fin (ny_fin)
202
203   if isDual == 1 do   # make all four fins
204       CALL("3164-05_fin.udo", px_fin, 0, 0, 0, 0, 6); # create the positive-x fin (px_fin)
205       CALL("3164-05_fin.udo", nx_fin, 180, 0, 0, 0, 6); # create the negative-x fin ↵
      (nx_fin)
206   endif;
207
```

-4-

Lindgren3164-05.udo 9/12/2007

```
208   # make NTF-box
209   ncell = int(max_xy/cellsize);
210   ntf_xy = 2*(ncell+4)*cellsize;
211   ntf_abs_xy = 2*(ncell+8)*cellsize;
212
213   ncell = int((max_z-min_z)/cellsize/2);
214   ntf_z = 2*(ncell+4)*cellsize;
215   ntf_abs_z = 2*(ncell+8)*cellsize;
216   ntf_z_pos = (max_z+min_z)/2;
217
218   CALL("boxes/ntf.udo", ntf_abs_xy, ntf_abs_xy, ntf_abs_z, ntf_xy, ntf_xy, ntf_z, N, N,
      N, x-(ntf_abs_xy/2), y-(ntf_abs_xy/2), z+ntf_z_pos-(ntf_abs_z/2), 13);
219
220   # add port and configure excitation
221   CALL("ports/point.udo", "fin_y_"@portf_y@"_o", 1, portf_y, 1, 1, x, y, z, 9);
222   CALL("ports/point.udo", "fin_y_"@portf_y@"_ny", 1, portf_y, 0, 0, x, y-fin_gap, z, 9);
223
224   if isDual == 1 do # if dual polarized, excite/terminate the second fin
225       CALL("ports/point.udo", "fin_x_"@portf_x@"_o", 1, portf_x, 1, 1, x, y, z, 9);
226       CALL("ports/point.udo", "fin_x_"@portf_x@"_nx", 1, portf_x, 0, 0, x-fin_gap, y, z,
      9);
227   endif;
228
229   # postprocessing
230   if fstep == -1 do
231       fstep = (fmax-fmin)/15;
232   endif;
233   SK1DIFF(fmin, fmax, fstep, 1);
234   NTF(ntf_freqs);
235
236
237   MARKFJ(ELEM, ALL, RESET);
238   MARK(ELEM, ALL, RESET);
239
240   CLOSEOBJ; #oname
241
```

3164-05_fin.udo

3164-05_fin.udo 9/12/2007

```
1   comment="creates a single fin from the Lindgren 3164-05 antenna using hardware
    dimensions from the Inventor files (BP).  Rotation is counter-clockwise about the
    z-axis, starting on the x-axis";
2   bitmap="nobitmap.bmp";
3   # Gabe Cohn, 07-09-2007
4
5   PAR("object name",oname,"fin");
6   PAR("Rotation",rot,0);
7
8   ENDHEADER;
9
10  TEST(rot >= 0 && rot < 360, "Rotation must be in range [0, 360)");
11
12  OPENOBJECT(oname);
13
14  # generate fin using consecutive layers of combined elements starting at the bottom
15
16  fin_top = 152.451;
17  fin_bot = -10.472;
18  fin_width = 3.615;
19  fin_w2 = fin_width / 2;
20  buff = 0.00002; # add to level values before calling MESHINDEX to be sure to include
    the desired level
21
22  if SLICINGPHASE == 1 dc # if in final phase
23      top_index = MESHINDEX(z+fin_top+buff);
24      TEST(top_index > 0, "Mesh not ready for FINAL phase drawing");
25      cur_index = MESHINDEX(z+fin_bot+buff); # start at the bottom of the fin
26      if cur_index < 0 dc
27          cur_index = 0;
28      endif;
29      cur_fx = 3.510;
30      cur_bx = 26.949;
31  endif; # final phase
32
33  # fin sector 1
34
35  # params
36  bot_z = -10.472; # level (z position) of bottom segment
37  bot_fx = 3.510; # x position of front edge of bottom segment
38  bot_bx = 26.949; # x position of back edge of bottom segment
39  top_z = -0.228; # level (z position) of top segment
40  top_fx = 3.510; # x position of front edge of top segment
41  top_bx = 26.949; # x position of back edge of top segment
42  f_angle = 90; # angle (up from x-axis) of the front edge
43  b_angle = 90; # angle (up from x-axis) of the back edge
44
45  if SLICINGPHASE == 0 dc # if in draft phase
46      ELEMENT(z+bot_z, 0, 15, metal, fin_1_b, INn); # bottom element
47          NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
48          ADDY(fin_width);
49          ADDLINE(x+bot_fx, y+fin_w2);
50          CLOSELINE;
51      ENDELEM;
```

3164-05_fin.udo 9/12/2007

```
52        ELEMENT(z+top_z, 0, 16, metal, fin_1_c, INn); # cover (top) element
53            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
54            ADDY(fin_width);
55            ADDLINE(x+top_fx, y+fin_w2);
56            CLOSELINE;
57        ENDELEM;
58    endif; # draft phase
59
60    if SLICINGPHASE == 1 do # if in final phase
61        top_index = MESHINDEX(z+top_z+buff);
62        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
      never happen, because the mesh should already be setup
63        OPENOBJECT(fin_1);
64            count = 1;
65            while cur_index < top_index do
66                bot_level = MESH(cur_index);
67                top_level = MESH(++cur_index);
68                if bot_level == (z+bot_z) do
69                    cur_fx = bot_fx;
70                    cur_bx = bot_bx;
71                endif;
72                ELEMENT(bot_level, 0, 15, metal, fin_1@"_"@count@b, INn); # bottom element
73                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
74                    ADDY(fin_width);
75                    ADDLINE(cur_fx, y+fin_w2);
76                    CLOSELINE;
77                ENDELEM;
78                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
79                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
80                ELEMENT(top_level, 0, 16, metal, fin_1@"_"@count@c, INn); # cover (top)
      element
81                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
82                    ADDY(fin_width);
83                    ADDLINE(cur_fx, y+fin_w2);
84                    CLOSELINE;
85                ENDELEM;
86                count++;
87            endwhile;
88        CLOSEOBJ; # fin_1
89    endif; # final phase
90
91    # fin sector 2b
92
93    # params
94    bot_z = -0.228; # level (z position) of bottom segment
95    bot_fx = 0.686; # x position of front edge of bottom segment
96    bot_bx = 26.949; # x position of back edge of bottom segment
97    top_z = 9.906; # level (z position) of top segment
98    top_fx = 0.686; # x position of front edge of top segment
99    top_bx = 26.949; # x position of back edge of top segment
100   f_angle = 90; # angle (up from x-axis) of the front edge
101   b_angle = 90; # angle (up from x-axis) of the back edge
102
103   if SLICINGPHASE == 0 do # if in draft phase
```

-2-

3164-05_fin.udo 9/12/2007

```
104         ELEMENT(z+bot_z, 0, 15, metal, fin_2b_b, INn); # bottom element
105             NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
106             ADDY(fin_width);
107             ADDLINE(x+bot_fx, y+fin_w2);
108             CLOSELINE;
109         ENDELEM;
110         ELEMENT(z+top_z, 0, 16, metal, fin_2b_c, INn); # cover (top) element
111             NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
112             ADDY(fin_width);
113             ADDLINE(x+top_fx, y+fin_w2);
114             CLOSELINE;
115         ENDELEM;
116     endif; # draft phase
117
118     if SLICINGPHASE == 1 dc # if in final phase
119         top_index = MESHINDEX(z+top_z+buff);
120         TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
121         OPENOBJECT(fin_2b);
122             count = 1;
123             while cur_index < top_index do
124                 bot_level = MESH(cur_index);
125                 top_level = MESH(++cur_index);
126                 if bot_level == (z+bot_z) do
127                     cur_fx = bot_fx;
128                     cur_bx = bot_bx;
129                 endif;
130                 ELEMENT(bot_level, 0, 15, metal, fin_2b@"_"@count@b, INn); # bottom element
131                     NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
132                     ADDY(fin_width);
133                     ADDLINE(cur_fx, y+fin_w2);
134                     CLOSELINE;
135                 ENDELEM;
136                 cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
137                 cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
138                 ELEMENT(top_level, 0, 16, metal, fin_2b@"_"@count@c, INn); # cover (top)
    element
139                     NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
140                     ADDY(fin_width);
141                     ADDLINE(cur_fx, y+fin_w2);
142                     CLOSELINE;
143                 ENDELEM;
144                 count++;
145             endwhile;
146         CLOSEOBJ; # fin_2b
147     endif; # final phase
148
149     # fin sector 3b
150
151     # params
152     bot_z = 9.906; # level (z position) of bottom segment
153     bot_fx = 0.686; # x position of front edge of bottom segment
154     bot_bx = 26.949; # x position of back edge of bottom segment
155     top_z = 20.079; # level (z position) of top segment
```

-3-

3164-05_fin.udo 9/12/2007

```
156   top_fx = 0.896; # x position of front edge of top segment
157   top_bx = 26.949; # x position of back edge of top segment
158   f_angle = 88.82; # angle (up from x-axis) of the front edge
159   b_angle = 90; # angle (up from x-axis) of the back edge
160
161   if SLICINGPHASE == 0 do # if in draft phase
162       ELEMENT(z+bot_z, 0, 15, metal, fin_3b_b, INn); # bottom element
163           NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
164           ADDY(fin_width);
165           ADDLINE(x+bot_fx, y+fin_w2);
166           CLOSELINE;
167       ENDELEM;
168       ELEMENT(z+top_z, 0, 16, metal, fin_3b_c, INn); # cover (top) element
169           NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
170           ADDY(fin_width);
171           ADDLINE(x+top_fx, y+fin_w2);
172           CLOSELINE;
173       ENDELEM;
174   endif; # draft phase
175
176   if SLICINGPHASE == 1 do # if in final phase
177       top_index = MESHINDEX(z+top_z+buff);
178       TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
      never happen, because the mesh should already be setup
179       OPENOBJECT(fin_3b);
180           count = 1;
181           while cur_index < top_index do
182               bot_level = MESH(cur_index);
183               top_level = MESH(++cur_index);
184               if bot_level == (z+bot_z) do
185                   cur_fx = bot_fx;
186                   cur_bx = bot_bx;
187               endif;
188               ELEMENT(bot_level, 0, 15, metal, fin_3b@"_"@count@b, INn); # bottom element
189                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
190                   ADDY(fin_width);
191                   ADDLINE(cur_fx, y+fin_w2);
192                   CLOSELINE;
193               ENDELEM;
194               cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
195               cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
196               ELEMENT(top_level, 0, 16, metal, fin_3b@"_"@count@c, INn); # cover (top)
      element
197                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
198                   ADDY(fin_width);
199                   ADDLINE(cur_fx, y+fin_w2);
200                   CLOSELINE;
201               ENDELEM;
202               count++;
203           endwhile;
204       CLOSEOBJ; # fin_3b
205   endif; # final phase
206
207   # fin sector 4b
```

-4-

3164-05_fin.udo 9/12/2007

```
208
209   # params
210   bot_z = 20.079; # level (z position) of bottom segment
211   bot_fx = 0.896; # x position of front edge of bottom segment
212   bot_bx = 26.949; # x position of back edge of bottom segment
213   top_z = 30.251; # level (z position) of top segment
214   top_fx = 1.092; # x position of front edge of top segment
215   top_bx = 30.884; # x position of back edge of top segment
216   f_angle = 88.82; # angle (up from x-axis) of the front edge
217   b_angle = 68.85; # angle (up from x-axis) of the back edge
218
219   if SLICINGPHASE == 0 do # if in draft phase
220       ELEMENT(z+bot_z, 0, 15, metal, fin_4b_b, INn); # bottom element
221           NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
222           ADDY(fin_width);
223           ADDLINE(x+bot_fx, y+fin_w2);
224           CLOSELINE;
225       ENDELEM;
226       ELEMENT(z+top_z, 0, 16, metal, fin_4b_c, INn); # cover (top) element
227           NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
228           ADDY(fin_width);
229           ADDLINE(x+top_fx, y+fin_w2);
230           CLOSELINE;
231       ENDELEM;
232   endif; # draft phase
233
234   if SLICINGPHASE == 1 do # if in final phase
235       top_index = MESHINDEX(z+top_z+buff);
236       TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
      never happen, because the mesh should already be setup
237       OPENOBJECT(fin_4b);
238           count = 1;
239           while cur_index < top_index do
240               bot_level = MESH(cur_index);
241               top_level = MESH(++cur_index);
242               if bot_level == (z+bot_z) do
243                   cur_fx = bot_fx;
244                   cur_bx = bot_bx;
245               endif;
246               ELEMENT(bot_level, 0, 15, metal, fin_4b@"_"@count@b, INn); # bottom element
247                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
248                   ADDY(fin_width);
249                   ADDLINE(cur_fx, y+fin_w2);
250                   CLOSELINE;
251               ENDELEM;
252               cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
253               cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
254               ELEMENT(top_level, 0, 16, metal, fin_4b@"_"@count@c, INn); # cover (top)
      element
255                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
256                   ADDY(fin_width);
257                   ADDLINE(cur_fx, y+fin_w2);
258                   CLOSELINE;
259               ENDELEM;
```

-5-

3164-05_fin.udo 9/12/2007

```
260             count++;
261         endwhile;
262     CLOSEOBJ; # fin_4b
263 endif; # final phase
264
265 # fin sector 5b
266
267 # params
268 bot_z = 30.251; # level (z position) of bottom segment
269 bot_fx = 1.092; # x position of front edge of bottom segment
270 bot_bx = 30.884; # x position of back edge of bottom segment
271 top_z = 40.437; # level (z position) of top segment
272 top_fx = 1.448; # x position of front edge of top segment
273 top_bx = 34.825; # x position of back edge of top segment
274 f_angle = 88.05; # angle (up from x-axis) of the front edge
275 b_angle = 68.85; # angle (up from x-axis) of the back edge
276
277 if SLICINGPHASE == 0 do # if in draft phase
278     ELEMENT(z+bot_z, 0, 15, metal, fin_5b_b, INn); # bottom element
279         NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
280         ADDY(fin_width);
281         ADDLINE(x+bot_fx, y+fin_w2);
282         CLOSELINE;
283     ENDELEM;
284     ELEMENT(z+top_z, 0, 16, metal, fin_5b_c, INn); # cover (top) element
285         NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
286         ADDY(fin_width);
287         ADDLINE(x+top_fx, y+fin_w2);
288         CLOSELINE;
289     ENDELEM;
290 endif; # draft phase
291
292 if SLICINGPHASE == 1 do # if in final phase
293     top_index = MESHINDEX(z+top_z+buff);
294     TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
295     OPENOBJECT(fin_5b);
296         count = 1;
297         while cur_index < top_index do
298             bot_level = MESH(cur_index);
299             top_level = MESH(++cur_index);
300             if bot_level == (z+bot_z) do
301                 cur_fx = bot_fx;
302                 cur_bx = bot_bx;
303             endif;
304             ELEMENT(bot_level, 0, 15, metal, fin_5b@"_"@count@b, INn); # bottom element
305                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
306                 ADDY(fin_width);
307                 ADDLINE(cur_fx, y+fin_w2);
308                 CLOSELINE;
309             ENDELEM;
310             cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
311             cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
312             ELEMENT(top_level, 0, 16, metal, fin_5b@"_"@count@c, INn); # cover (top)
```

-6-

3164-05_fin.udo **9/12/2007**

```
element
                NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
                ADDY(fin_width);
                ADDLINE(cur_fx, y+fin_w2);
                CLOSELINE;
            ENDELEM;
            count++;
        endwhile;
    CLOSEOBJ; # fin_5b
endif; # final phase

# fin sector 6b

# params
bot_z = 40.437; # level (z position) of bottom segment
bot_fx = 1.448; # x position of front edge of bottom segment
bot_bx = 34.825; # x position of back edge of bottom segment
top_z = 45.415; # level (z position) of top segment
top_fx = 1.803; # x position of front edge of top segment
top_bx = 36.751; # x position of back edge of top segment
f_angle = 85.89; # angle (up from x-axis) of the front edge
b_angle = 68.85; # angle (up from x-axis) of the back edge

if SLICINGPHASE == 0 do # if in draft phase
    ELEMENT(z+bot_z, 0, 15, metal, fin_6b_b, INn); # bottom element
        NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
        ADDY(fin_width);
        ADDLINE(x+bot_fx, y+fin_w2);
        CLOSELINE;
    ENDELEM;
    ELEMENT(z+top_z, 0, 16, metal, fin_6b_c, INn); # cover (top) element
        NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
        ADDY(fin_width);
        ADDLINE(x+top_fx, y+fin_w2);
        CLOSELINE;
    ENDELEM;
endif; # draft phase

if SLICINGPHASE == 1 do # if in final phase
    top_index = MESHINDEX(z+top_z+buff);
    TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
never happen, because the mesh should already be setup
    OPENOBJECT(fin_6b);
        count = 1;
        while cur_index < top_index do
            bot_level = MESH(cur_index);
            top_level = MESH(++cur_index);
            if bot_level == (z+bot_z) do
                cur_fx = bot_fx;
                cur_bx = bot_bx;
            endif;
            ELEMENT(bot_level, 0, 15, metal, fin_6b@"_"@count@b, INn); # bottom element
                NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
                ADDY(fin_width);
```

3164-05_fin.udo 9/12/2007

```
365                 ADDLINE(cur_fx, y+fin_w2);
366                 CLOSELINE;
367             ENDELEM;
368             cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
369             cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
370             ELEMENT(top_level, 0, 16, metal, fin_6b@"_"@count@c, INn); # cover (top) element
371                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
372                 ADDY(fin_width);
373                 ADDLINE(cur_fx, y+fin_w2);
374                 CLOSELINE;
375             ENDELEM;
376             count++;
377         endwhile;
378     CLOSEOBJ; # fin_6b
379 endif; # final phase
380
381 # fin sector 7b
382
383 # params
384 bot_z = 45.415; # level (z position) of bottom segment
385 bot_fx = 1.803; # x position of front edge of bottom segment
386 bot_bx = 36.751; # x position of back edge of bottom segment
387 top_z = 50.622; # level (z position) of top segment
388 top_fx = 2.413; # x position of front edge of top segment
389 top_bx = 38.765; # x position of back edge of top segment
390 f_angle = 83.34; # angle (up from x-axis) of the front edge
391 b_angle = 68.85; # angle (up from x-axis) of the back edge
392
393 if SLICINGPHASE == 0 do # if in draft phase
394     ELEMENT(z+bot_z, 0, 15, metal, fin_7b_b, INn); # bottom element
395         NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
396         ADDY(fin_width);
397         ADDLINE(x+bot_fx, y+fin_w2);
398         CLOSELINE;
399     ENDELEM;
400     ELEMENT(z+top_z, 0, 16, metal, fin_7b_c, INn); # cover (top) element
401         NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
402         ADDY(fin_width);
403         ADDLINE(x+top_fx, y+fin_w2);
404         CLOSELINE;
405     ENDELEM;
406 endif; # draft phase
407
408 if SLICINGPHASE == 1 do # if in final phase
409     top_index = MESHINDEX(z+top_z+buff);
410     TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should never happen, because the mesh should already be setup
411     OPENOBJECT(fin_7b);
412         count = 1;
413         while cur_index < top_index do
414             bot_level = MESH(cur_index);
415             top_level = MESH(++cur_index);
416             if bot_level == (z+bot_z) do
```

-8-

```
417                 cur_fx = bot_fx;
418                 cur_bx = bot_bx;
419             endif;
420             ELEMENT(bot_level, 0, 15, metal, fin_7b@"_"@count@b, INn); # bottom element
421                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
422                 ADDY(fin_width);
423                 ADDLINE(cur_fx, y+fin_w2);
424                 CLOSELINE;
425             ENDELEM;
426             cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
427             cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
428             ELEMENT(top_level, 0, 16, metal, fin_7b@"_"@count@c, INn); # cover (top) element
429                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
430                 ADDY(fin_width);
431                 ADDLINE(cur_fx, y+fin_w2);
432                 CLOSELINE;
433             ENDELEM;
434             count++;
435         endwhile;
436     CLOSEOBJ; # fin_7b
437 endif; # final phase
438
439 # fin sector 8b
440
441 # params
442 bot_z = 50.622; # level (z position) of bottom segment
443 bot_fx = 2.413; # x position of front edge of bottom segment
444 bot_bx = 38.765; # x position of back edge of bottom segment
445 top_z = 60.808; # level (z position) of top segment
446 top_fx = 3.404; # x position of front edge of top segment
447 top_bx = 42.706; # x position of back edge of top segment
448 f_angle = 84.52; # angle (up from x-axis) of the front edge
449 b_angle = 68.85; # angle (up from x-axis) of the back edge
450
451 if SLICINGPHASE == 0 do # if in draft phase
452     ELEMENT(z+bot_z, 0, 15, metal, fin_8b_b, INn); # bottom element
453         NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
454         ADDY(fin_width);
455         ADDLINE(x+bot_fx, y+fin_w2);
456         CLOSELINE;
457     ENDELEM;
458     ELEMENT(z+top_z, 0, 16, metal, fin_8b_c, INn); # cover (top) element
459         NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
460         ADDY(fin_width);
461         ADDLINE(x+top_fx, y+fin_w2);
462         CLOSELINE;
463     ENDELEM;
464 endif; # draft phase
465
466 if SLICINGPHASE == 1 do # if in final phase
467     top_index = MESHINDEX(z+top_z+buff);
468     TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should never happen, because the mesh should already be setup
```

```
469       OPENOBJECT(fin_8b);
470           count = 1;
471           while cur_index < top_index do
472               bot_level = MESH(cur_index);
473               top_level = MESH(++cur_index);
474               if bot_level == (z+bot_z) do
475                   cur_fx = bot_fx;
476                   cur_bx = bot_bx;
477               endif;
478               ELEMENT(bot_level, 0, 15, metal, fin_8b@"_"@count@b, INn); # bottom element
479                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
480                   ADDY(fin_width);
481                   ADDLINE(cur_fx, y+fin_w2);
482                   CLOSELINE;
483               ENDELEM;
484               cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
485               cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
486               ELEMENT(top_level, 0, 16, metal, fin_8b@"_"@count@c, INn); # cover (top) ↵
    element
487                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
488                   ADDY(fin_width);
489                   ADDLINE(cur_fx, y+fin_w2);
490                   CLOSELINE;
491               ENDELEM;
492               count++;
493           endwhile;
494       CLOSEOBJ; # fin_8b
495   endif; # final phase
496
497   # fin sector 9b
498
499   # params
500   bot_z = 60.808; # level (z position) of bottom segment
501   bot_fx = 3.404; # x position of front edge of bottom segment
502   bot_bx = 42.706; # x pcsition of back edge of bottom segment
503   top_z = 70.993; # level (z position) of top segment
504   top_fx = 4.750; # x position of front edge of top segment
505   top_bx = 46.646; # x pcsition of back edge of top segment
506   f_angle = 82.37; # angle (up from x-axis) of the front edge
507   b_angle = 68.85; # angle (up from x-axis) of the back edge
508
509   if SLICINGPHASE == 0 dc # if in draft phase
510       ELEMENT(z+bot_z, 0, 15, metal, fin_9b_b, INn); # bottom element
511           NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
512           ADDY(fin_width);
513           ADDLINE(x+bot_fx, y+fin_w2);
514           CLOSELINE;
515       ENDELEM;
516       ELEMENT(z+top_z, 0, 16, metal, fin_9b_c, INn); # cover (top) element
517           NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
518           ADDY(fin_width);
519           ADDLINE(x+top_fx, y+fin_w2);
520           CLOSELINE;
521       ENDELEM;
```

```
522   endif; # draft phase
523
524   if SLICINGPHASE == 1 do # if in final phase
525       top_index = MESHINDEX(z+top_z+buff);
526       TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
      never happen, because the mesh should already be setup
527       OPENOBJECT(fin_9b);
528           count = 1;
529           while cur_index < top_index do
530               bot_level = MESH(cur_index);
531               top_level = MESH(++cur_index);
532               if bot_level == (z+bot_z) do
533                   cur_fx = bot_fx;
534                   cur_bx = bot_bx;
535               endif;
536               ELEMENT(bot_level, 0, 15, metal, fin_9b@"_"@count@b, INn); # bottom element
537                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
538                   ADDY(fin_width);
539                   ADDLINE(cur_fx, y+fin_w2);
540                   CLOSELINE;
541               ENDELEM;
542               cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
543               cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
544               ELEMENT(top_level, 0, 16, metal, fin_9b@"_"@count@c, INn); # cover (top)
      element
545                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
546                   ADDY(fin_width);
547                   ADDLINE(cur_fx, y+fin_w2);
548                   CLOSELINE;
549               ENDELEM;
550               count++;
551           endwhile;
552       CLOSEOBJ; # fin_9b
553   endif; # final phase
554
555   # fin sector 10
556
557   # params
558   bot_z = 70.993; # level (z position) of bottom segment
559   bot_fx = 4.750; # x position of front edge of bottom segment
560   bot_bx = 46.646; # x position of back edge of bottom segment
561   top_z = 81.178; # level (z position) of top segment
562   top_fx = 6.604; # x position of front edge of top segment
563   top_bx = 50.586; # x position of back edge of top segment
564   f_angle = 79.79; # angle (up from x-axis) of the front edge
565   b_angle = 68.85; # angle (up from x-axis) of the back edge
566
567   if SLICINGPHASE == 0 do # if in draft phase
568       ELEMENT(z+bot_z, 0, 15, metal, fin_10_b, INn); # bottom element
569           NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
570           ADDY(fin_width);
571           ADDLINE(x+bot_fx, y+fin_w2);
572           CLOSELINE;
573       ENDELEM;
```

3164-05_fin.udo 9/12/2007

```
574         ELEMENT(z+top_z, 0, 16, metal, fin_10_c, INn); # cover (top) element
575             NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
576             ADDY(fin_width);
577             ADDLINE(x+top_fx, y+fin_w2);
578             CLOSELINE;
579         ENDELEM;
580     endif; # draft phase
581
582     if SLICINGPHASE == 1 dc # if in final phase
583         top_index = MESHINDEX(z+top_z+buff);
584         TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
585         OPENOBJECT(fin_10);
586             count = 1;
587             while cur_index < top_index do
588                 bot_level = MESH(cur_index);
589                 top_level = MESH(++cur_index);
590                 if bot_level == (z+bot_z) do
591                     cur_fx = bot_fx;
592                     cur_bx = bot_bx;
593                 endif;
594                 ELEMENT(bot_level, 0, 15, metal, fin_10@"_"@count@b, INn); # bottom element
595                     NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
596                     ADDY(fin_width);
597                     ADDLINE(cur_fx, y+fin_w2);
598                     CLOSELINE;
599                 ENDELEM;
600                 cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
601                 cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
602                 ELEMENT(top_level, 0, 16, metal, fin_10@"_"@count@c, INn); # cover (top)
    element
603                     NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
604                     ADDY(fin_width);
605                     ADDLINE(cur_fx, y+fin_w2);
606                     CLOSELINE;
607                 ENDELEM;
608                 count++;
609             endwhile;
610         CLOSEOBJ; # fin_10
611     endif; # final phase
612
613     # fin sector 11
614
615     # params
616     bot_z = 81.178; # level (z position) of bottom segment
617     bot_fx = 6.604; # x position of front edge of bottom segment
618     bot_bx = 50.586; # x position of back edge of bottom segment
619     top_z = 91.364; # level (z position) of top segment
620     top_fx = 8.636; # x position of front edge of top segment
621     top_bx = 54.527; # x position of back edge of top segment
622     f_angle = 78.69; # angle (up from x-axis) of the front edge
623     b_angle = 68.85; # angle (up from x-axis) of the back edge
624
625     if SLICINGPHASE == 0 dc # if in draft phase
```

-12-

3164-05_fin.udo 9/12/2007

```
626         ELEMENT(z+bot_z, 0, 15, metal, fin_11_b, INn); # bottom element
627             NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
628             ADDY(fin_width);
629             ADDLINE(x+bot_fx, y+fin_w2);
630             CLOSELINE;
631         ENDELEM;
632         ELEMENT(z+top_z, 0, 16, metal, fin_11_c, INn); # cover (top) element
633             NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
634             ADDY(fin_width);
635             ADDLINE(x+top_fx, y+fin_w2);
636             CLOSELINE;
637         ENDELEM;
638     endif; # draft phase
639
640     if SLICINGPHASE == 1 dc # if in final phase
641         top_index = MESHINDEX(z+top_z+buff);
642         TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
643         OPENOBJECT(fin_11);
644             count = 1;
645             while cur_index < top_index do
646                 bot_level = MESH(cur_index);
647                 top_level = MESH(++cur_index);
648                 if bot_level == (z+bot_z) do
649                     cur_fx = bot_fx;
650                     cur_bx = bot_bx;
651                 endif;
652                 ELEMENT(bot_level, 0, 15, metal, fin_11@"_"@count@b, INn); # bottom element
653                     NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
654                     ADDY(fin_width);
655                     ADDLINE(cur_fx, y+fin_w2);
656                     CLOSELINE;
657                 ENDELEM;
658                 cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
659                 cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
660                 ELEMENT(top_level, 0, 16, metal, fin_11@"_"@count@c, INn); # cover (top)
    element
661                     NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
662                     ADDY(fin_width);
663                     ADDLINE(cur_fx, y+fin_w2);
664                     CLOSELINE;
665                 ENDELEM;
666                 count++;
667             endwhile;
668         CLOSEOBJ; # fin_11
669     endif; # final phase
670
671     # fin sector 12
672
673     # params
674     bot_z = 91.364; # level (z position) of bottom segment
675     bot_fx = 8.636; # x position of front edge of bottom segment
676     bot_bx = 54.527; # x position of back edge of bottom segment
677     top_z = 101.549; # level (z position) of top segment
```

-13-

3164-05_fin.udo 9/12/2007

```
678   top_fx = 11.684; # x position of front edge of top segment
679   top_bx = 58.467; # x position of back edge of top segment
680   f_angle = 73.31; # angle (up from x-axis) of the front edge
681   b_angle = 68.85; # angle (up from x-axis) of the back edge
682
683   if SLICINGPHASE == 0 do # if in draft phase
684       ELEMENT(z+bot_z, 0, 15, metal, fin_12_b, INn); # bottom element
685           NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
686           ADDY(fin_width);
687           ADDLINE(x+bot_fx, y+fin_w2);
688           CLOSELINE;
689       ENDELEM;
690       ELEMENT(z+top_z, 0, 16, metal, fin_12_c, INn); # cover (top) element
691           NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
692           ADDY(fin_width);
693           ADDLINE(x+top_fx, y+fin_w2);
694           CLOSELINE;
695       ENDELEM;
696   endif; # draft phase
697
698   if SLICINGPHASE == 1 do # if in final phase
699       top_index = MESHINDEX(z+top_z+buff);
700       TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
      never happen, because the mesh should already be setup
701       OPENOBJECT(fin_12);
702           count = 1;
703           while cur_index < top_index do
704               bot_level = MESH(cur_index);
705               top_level = MESH(++cur_index);
706               if bot_level == (z+bot_z) do
707                   cur_fx = bot_fx;
708                   cur_bx = bot_bx;
709               endif;
710               ELEMENT(bot_level, 0, 15, metal, fin_12@"_"@count@b, INn); # bottom element
711                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
712                   ADDY(fin_width);
713                   ADDLINE(cur_fx, y+fin_w2);
714                   CLOSELINE;
715               ENDELEM;
716               cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
717               cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
718               ELEMENT(top_level, 0, 16, metal, fin_12@"_"@count@c, INn); # cover (top)
      element
719                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
720                   ADDY(fin_width);
721                   ADDLINE(cur_fx, y+fin_w2);
722                   CLOSELINE;
723               ENDELEM;
724               count++;
725           endwhile;
726       CLOSEOBJ; # fin_12
727   endif; # final phase
728
729   # fin sector 13
```

-14-

3164-05_fin.udo 9/12/2007

```
730
731   # params
732   bot_z = 101.549; # level (z position) of bottom segment
733   bot_fx = 11.684; # x position of front edge of bottom segment
734   bot_bx = 58.467; # x position of back edge of bottom segment
735   top_z = 121.920; # level (z position) of top segment
736   top_fx = 19.025; # x position of front edge of top segment
737   top_bx = 66.348; # x position of back edge of top segment
738   f_angle = 70.2; # angle (up from x-axis) of the front edge
739   b_angle = 68.85; # angle (up from x-axis) of the back edge
740
741   if SLICINGPHASE == 0 do # if in draft phase
742       ELEMENT(z+bot_z, 0, 15, metal, fin_13_b, INn); # bottom element
743           NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
744           ADDY(fin_width);
745           ADDLINE(x+bot_fx, y+fin_w2);
746           CLOSELINE;
747       ENDELEM;
748       ELEMENT(z+top_z, 0, 16, metal, fin_13_c, INn); # cover (top) element
749           NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
750           ADDY(fin_width);
751           ADDLINE(x+top_fx, y+fin_w2);
752           CLOSELINE;
753       ENDELEM;
754   endif; # draft phase
755
756   if SLICINGPHASE == 1 do # if in final phase
757       top_index = MESHINDEX(z+top_z+buff);
758       TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
      never happen, because the mesh should already be setup
759       OPENOBJECT(fin_13);
760           count = 1;
761           while cur_index < top_index do
762               bot_level = MESH(cur_index);
763               top_level = MESH(++cur_index);
764               if bot_level == (z+bot_z) do
765                   cur_fx = bot_fx;
766                   cur_bx = bot_bx;
767               endif;
768               ELEMENT(bot_level, 0, 15, metal, fin_13@"_"@count@b, INn); # bottom element
769                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
770                   ADDY(fin_width);
771                   ADDLINE(cur_fx, y+fin_w2);
772                   CLOSELINE;
773               ENDELEM;
774               cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
775               cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
776               ELEMENT(top_level, 0, 16, metal, fin_13@"_"@count@c, INn); # cover (top)
      element
777                   NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
778                   ADDY(fin_width);
779                   ADDLINE(cur_fx, y+fin_w2);
780                   CLOSELINE;
781               ENDELEM;
```

-15-

3164-05_fin.udo 9/12/2007

```
782             count++;
783         endwhile;
784     CLOSEOBJ; # fin_13
785 endif; # final phase
786
787 # fin sector 14
788
789 # params
790 bot_z = 121.920; # level (z position) of bottom segment
791 bot_fx = 19.025; # x position of front edge of bottom segment
792 bot_bx = 66.348; # x position of back edge of bottom segment
793 top_z = 129.540; # level (z position) of top segment
794 top_fx = 22.073; # x position of front edge of top segment
795 top_bx = 69.296; # x position of back edge of top segment
796 f_angle = 68.15; # angle (up from x-axis) of the front edge
797 b_angle = 68.85; # angle (up from x-axis) of the back edge
798
799 if SLICINGPHASE == 0 do # if in draft phase
800     ELEMENT(z+bot_z, 0, 15, metal, fin_14_b, INn); # bottom element
801         NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
802         ADDY(fin_width);
803         ADDLINE(x+bot_fx, y+fin_w2);
804         CLOSELINE;
805     ENDELEM;
806     ELEMENT(z+top_z, 0, 16, metal, fin_14_c, INn); # cover (top) element
807         NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
808         ADDY(fin_width);
809         ADDLINE(x+top_fx, y+fin_w2);
810         CLOSELINE;
811     ENDELEM;
812 endif; # draft phase
813
814 if SLICINGPHASE == 1 do # if in final phase
815     top_index = MESHINDEX(z+top_z+buff);
816     TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
817     OPENOBJECT(fin_14);
818         count = 1;
819         while cur_index < top_index do
820             bot_level = MESH(cur_index);
821             top_level = MESH(++cur_index);
822             if bot_level == (z+bot_z) do
823                 cur_fx = bot_fx;
824                 cur_bx = bot_bx;
825             endif;
826             ELEMENT(bot_level, 0, 15, metal, fin_14@"_"@count@b, INn); # bottom element
827                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
828                 ADDY(fin_width);
829                 ADDLINE(cur_fx, y+fin_w2);
830                 CLOSELINE;
831             ENDELEM;
832             cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
833             cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
834             ELEMENT(top_level, 0, 16, metal, fin_14@"_"@count@c, INn); # cover (top)
```

-16-

3164-05_fin.udo 9/12/2007

```
    element
                NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
                ADDY(fin_width);
                ADDLINE(cur_fx, y+fin_w2);
                CLOSELINE;
            ENDELEM;
            count++;
        endwhile;
    CLOSEOBJ; # fin_14
endif; # final phase

# fin sector 15

# params
bot_z = 129.540; # level (z position) of bottom segment
bot_fx = 22.073; # x position of front edge of bottom segment
bot_bx = 69.296; # x position of back edge of bottom segment
top_z = 137.185; # level (z position) of top segment
top_fx = 26.137; # x position of front edge of top segment
top_bx = 72.254; # x position of back edge of top segment
f_angle = 61.99; # angle (up from x-axis) of the front edge
b_angle = 68.85; # angle (up from x-axis) of the back edge

if SLICINGPHASE == 0 do # if in draft phase
    ELEMENT(z+bot_z, 0, 15, metal, fin_15_b, INn); # bottom element
        NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
        ADDY(fin_width);
        ADDLINE(x+bot_fx, y+fin_w2);
        CLOSELINE;
    ENDELEM;
    ELEMENT(z+top_z, 0, 16, metal, fin_15_c, INn); # cover (top) element
        NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
        ADDY(fin_width);
        ADDLINE(x+top_fx, y+fin_w2);
        CLOSELINE;
    ENDELEM;
endif; # draft phase

if SLICINGPHASE == 1 do # if in final phase
    top_index = MESHINDEX(z+top_z+buff);
    TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
never happen, because the mesh should already be setup
    OPENOBJECT(fin_15);
        count = 1;
        while cur_index < top_index do
            bot_level = MESH(cur_index);
            top_level = MESH(++cur_index);
            if bot_level == (z+bot_z) do
                cur_fx = bot_fx;
                cur_bx = bot_bx;
            endif;
            ELEMENT(bot_level, 0, 15, metal, fin_15@"_"@count@b, INn); # bottom element
                NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
                ADDY(fin_width);
```

-17-

3164-05_fin.udo 9/12/2007

```
887                 ADDLINE(cur_fx, y+fin_w2);
888                 CLOSELINE;
889             ENDELEM;
890             cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
891             cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
892             ELEMENT(top_level, 0, 16, metal, fin_15@"_"@count@c, INn); # cover (top) element
893                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
894                 ADDY(fin_width);
895                 ADDLINE(cur_fx, y+fin_w2);
896                 CLOSELINE;
897             ENDELEM;
898             count++;
899         endwhile;
900     CLOSEOBJ; # fin_15
901 endif; # final phase
902
903 # fin sector 16
904
905 # params
906 bot_z = 137.185; # level (z position) of bottom segment
907 bot_fx = 26.137; # x position of front edge of bottom segment
908 bot_bx = 72.254; # x position of back edge of bottom segment
909 top_z = 140.995; # level (z position) of top segment
910 top_fx = 29.185; # x position of front edge of top segment
911 top_bx = 73.728; # x position of back edge of top segment
912 f_angle = 51.34; # angle (up from x-axis) of the front edge
913 b_angle = 68.85; # angle (up from x-axis) of the back edge
914
915 if SLICINGPHASE == 0 do # if in draft phase
916     ELEMENT(z+bot_z, 0, 15, metal, fin_16_b, INn); # bottom element
917         NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
918         ADDY(fin_width);
919         ADDLINE(x+bot_fx, y+fin_w2);
920         CLOSELINE;
921     ENDELEM;
922     ELEMENT(z+top_z, 0, 16, metal, fin_16_c, INn); # cover (top) element
923         NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
924         ADDY(fin_width);
925         ADDLINE(x+top_fx, y+fin_w2);
926         CLOSELINE;
927     ENDELEM;
928 endif; # draft phase
929
930 if SLICINGPHASE == 1 do # if in final phase
931     top_index = MESHINDEX(z+top_z+buff);
932     TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should never happen, because the mesh should already be setup
933     OPENOBJECT(fin_16);
934         count = 1;
935         while cur_index < top_index do
936             bot_level = MESH(cur_index);
937             top_level = MESH(++cur_index);
938             if bot_level == (z+bot_z) do
```

-18-

3164-05_fin.udo 9/12/2007

```
939                 cur_fx = bot_fx;
940                 cur_bx = bot_bx;
941             endif;
942             ELEMENT(bot_level, 0, 15, metal, fin_16@"_"@count@b, INn); # bottom element
943                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
944                 ADDY(fin_width);
945                 ADDLINE(cur_fx, y+fin_w2);
946                 CLOSELINE;
947             ENDELEM;
948             cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
949             cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
950             ELEMENT(top_level, 0, 16, metal, fin_16@"_"@count@c, INn); # cover (top) element
951                 NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
952                 ADDY(fin_width);
953                 ADDLINE(cur_fx, y+fin_w2);
954                 CLOSELINE;
955             ENDELEM;
956             count++;
957         endwhile;
958     CLOSEOBJ; # fin_16
959 endif; # final phase
960
961 # fin sector 17
962
963 # params
964 bot_z = 140.995; # level (z position) of bottom segment
965 bot_fx = 29.185; # x position of front edge of bottom segment
966 bot_bx = 73.728; # x position of back edge of bottom segment
967 top_z = 142.265; # level (z position) of top segment
968 top_fx = 30.405; # x position of front edge of top segment
969 top_bx = 74.219; # x position of back edge of top segment
970 f_angle = 46.16; # angle (up from x-axis) of the front edge
971 b_angle = 68.85; # angle (up from x-axis) of the back edge
972
973 if SLICINGPHASE == 0 do # if in draft phase
974     ELEMENT(z+bot_z, 0, 15, metal, fin_17_b, INn); # bottom element
975         NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
976         ADDY(fin_width);
977         ADDLINE(x+bot_fx, y+fin_w2);
978         CLOSELINE;
979     ENDELEM;
980     ELEMENT(z+top_z, 0, 16, metal, fin_17_c, INn); # cover (top) element
981         NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
982         ADDY(fin_width);
983         ADDLINE(x+top_fx, y+fin_w2);
984         CLOSELINE;
985     ENDELEM;
986 endif; # draft phase
987
988 if SLICINGPHASE == 1 do # if in final phase
989     top_index = MESHINDEX(z+top_z+buff);
990     TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should never happen, because the mesh should already be setup
```

-19-

3164-05_fin.udo **9/12/2007**

```
991         OPENOBJECT(fin_17);
992             count = 1;
993             while cur_index < top_index do
994                 bot_level = MESH(cur_index);
995                 top_level = MESH(++cur_index);
996                 if bot_level == (z+bot_z) do
997                     cur_fx = bot_fx;
998                     cur_bx = bot_bx;
999                 endif;
1000                ELEMENT(bot_level, 0, 15, metal, fin_17@"_"@count@b, INn); # bottom element
1001                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1002                    ADDY(fin_width);
1003                    ADDLINE(cur_fx, y+fin_w2);
1004                    CLOSELINE;
1005                ENDELEM;
1006                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1007                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1008                ELEMENT(top_level, 0, 16, metal, fin_17@"_"@count@c, INn); # cover (top) ↵
    element
1009                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1010                    ADDY(fin_width);
1011                    ADDLINE(cur_fx, y+fin_w2);
1012                    CLOSELINE;
1013                ENDELEM;
1014                count++;
1015            endwhile;
1016        CLOSEOBJ; # fin_17
1017    endif; # final phase
1018
1019    # fin sector 18
1020
1021    # params
1022    bot_z = 142.265; # level (z position) of bottom segment
1023    bot_fx = 30.405; # x position of front edge of bottom segment
1024    bot_bx = 74.219; # x position of back edge of bottom segment
1025    top_z = 144.831; # level (z position) of top segment
1026    top_fx = 32.868; # x position of front edge of top segment
1027    top_bx = 73.812; # x position of back edge of top segment
1028    f_angle = 46.16; # angle (up from x-axis) of the front edge
1029    b_angle = 99; # angle (up from x-axis) of the back edge
1030
1031    if SLICINGPHASE == 0 dc # if in draft phase
1032        ELEMENT(z+bot_z, 0, 15, metal, fin_18_b, INn); # bottom element
1033            NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
1034            ADDY(fin_width);
1035            ADDLINE(x+bot_fx, y+fin_w2);
1036            CLOSELINE;
1037        ENDELEM;
1038        ELEMENT(z+top_z, 0, 16, metal, fin_18_c, INn); # cover (top) element
1039            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
1040            ADDY(fin_width);
1041            ADDLINE(x+top_fx, y+fin_w2);
1042            CLOSELINE;
1043        ENDELEM;
```

3164-05_fin.udo 9/12/2007

```
1044    endif; # draft phase
1045
1046    if SLICINGPHASE == 1 dc # if in final phase
1047        top_index = MESHINDEX(z+top_z+buff);
1048        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
        never happen, because the mesh should already be setup
1049        OPENOBJECT(fin_18);
1050            count = 1;
1051            while cur_index < top_index do
1052                bot_level = MESH(cur_index);
1053                top_level = MESH(++cur_index);
1054                if bot_level == (z+bot_z) do
1055                    cur_fx = bot_fx;
1056                    cur_bx = bot_bx;
1057                endif;
1058                ELEMENT(bot_level, 0, 15, metal, fin_18@"_"@count@b, INn); # bottom element
1059                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1060                    ADDY(fin_width);
1061                    ADDLINE(cur_fx, y+fin_w2);
1062                    CLOSELINE;
1063                ENDELEM;
1064                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1065                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1066                ELEMENT(top_level, 0, 16, metal, fin_18@"_"@count@c, INn); # cover (top)
        element
1067                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1068                    ADDY(fin_width);
1069                    ADDLINE(cur_fx, y+fin_w2);
1070                    CLOSELINE;
1071                ENDELEM;
1072                count++;
1073            endwhile;
1074        CLOSEOBJ; # fin_18
1075    endif; # final phase
1076
1077    # fin sector 19
1078
1079    # params
1080    bot_z = 144.831; # level (z position) of bottom segment
1081    bot_fx = 32.868; # x position of front edge of bottom segment
1082    bot_bx = 73.812; # x position of back edge of bottom segment
1083    top_z = 147.371; # level (z position) of top segment
1084    top_fx = 35.578; # x position of front edge of top segment
1085    top_bx = 73.279; # x position of back edge of top segment
1086    f_angle = 43.15; # angle (up from x-axis) of the front edge
1087    b_angle = 101.86; # angle (up from x-axis) of the back edge
1088
1089    if SLICINGPHASE == 0 dc # if in draft phase
1090        ELEMENT(z+bot_z, 0, 15, metal, fin_19_b, INn); # bottom element
1091            NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
1092            ADDY(fin_width);
1093            ADDLINE(x+bot_fx, y+fin_w2);
1094            CLOSELINE;
1095        ENDELEM;
```

-21-

3164-05_fin.udo 9/12/2007

```
1096        ELEMENT(z+top_z, 0, 16, metal, fin_19_c, INn); # cover (top) element
1097            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
1098            ADDY(fin_width);
1099            ADDLINE(x+top_fx, y+fin_w2);
1100            CLOSELINE;
1101        ENDELEM;
1102    endif; # draft phase
1103
1104    if SLICINGPHASE == 1 do # if in final phase
1105        top_index = MESHINDEX(z+top_z+buff);
1106        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
        never happen, because the mesh should already be setup
1107        OPENOBJECT(fin_19);
1108            count = 1;
1109            while cur_index < top_index do
1110                bot_level = MESH(cur_index);
1111                top_level = MESH(++cur_index);
1112                if bot_level == (z+bot_z) do
1113                    cur_fx = bot_fx;
1114                    cur_bx = bot_bx;
1115                endif;
1116                ELEMENT(bot_level, 0, 15, metal, fin_19@"_"@count@b, INn); # bottom element
1117                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1118                    ADDY(fin_width);
1119                    ADDLINE(cur_fx, y+fin_w2);
1120                    CLOSELINE;
1121                ENDELEM;
1122                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1123                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1124                ELEMENT(top_level, 0, 16, metal, fin_19@"_"@count@c, INn); # cover (top)
        element
1125                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1126                    ADDY(fin_width);
1127                    ADDLINE(cur_fx, y+fin_w2);
1128                    CLOSELINE;
1129                ENDELEM;
1130                count++;
1131            endwhile;
1132        CLOSEOBJ; # fin_19
1133    endif; # final phase
1134
1135    # fin sector 20
1136
1137    # params
1138    bot_z = 147.371; # level (z position) of bottom segment
1139    bot_fx = 35.578; # x position of front edge of bottom segment
1140    bot_bx = 73.279; # x position of back edge of bottom segment
1141    top_z = 148.641; # level (z position) of top segment
1142    top_fx = 36.932; # x position of front edge of top segment
1143    top_bx = 71.831; # x position of back edge of top segment
1144    f_angle = 43.15; # angle (up from x-axis) of the front edge
1145    b_angle = 138.74; # angle (up from x-axis) of the back edge
1146
1147    if SLICINGPHASE == 0 do # if in draft phase
```

-22-

3164-05_fin.udo 9/12/2007

```
1148        ELEMENT(z+bot_z, 0, 15, metal, fin_20_b, INn); # bottom element
1149            NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
1150            ADDY(fin_width);
1151            ADDLINE(x+bot_fx, y+fin_w2);
1152            CLOSELINE;
1153        ENDELEM;
1154        ELEMENT(z+top_z, 0, 16, metal, fin_20_c, INn); # cover (top) element
1155            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
1156            ADDY(fin_width);
1157            ADDLINE(x+top_fx, y+fin_w2);
1158            CLOSELINE;
1159        ENDELEM;
1160    endif; # draft phase
1161
1162    if SLICINGPHASE == 1 dc # if in final phase
1163        top_index = MESHINDEX(z+top_z+buff);
1164        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
1165        OPENOBJECT(fin_20);
1166            count = 1;
1167            while cur_index < top_index do
1168                bot_level = MESH(cur_index);
1169                top_level = MESH(++cur_index);
1170                if bot_level == (z+bot_z) do
1171                    cur_fx = bot_fx;
1172                    cur_bx = bot_bx;
1173                endif;
1174                ELEMENT(bot_level, 0, 15, metal, fin_20@"_"@count@b, INn); # bottom element
1175                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1176                    ADDY(fin_width);
1177                    ADDLINE(cur_fx, y+fin_w2);
1178                    CLOSELINE;
1179                ENDELEM;
1180                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1181                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1182                ELEMENT(top_level, 0, 16, metal, fin_20@"_"@count@c, INn); # cover (top)
    element
1183                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1184                    ADDY(fin_width);
1185                    ADDLINE(cur_fx, y+fin_w2);
1186                    CLOSELINE;
1187                ENDELEM;
1188                count++;
1189            endwhile;
1190        CLOSEOBJ; # fin_20
1191    endif; # final phase
1192
1193    # fin sector 21
1194
1195    # params
1196    bot_z = 148.641; # level (z position) of bottom segment
1197    bot_fx = 36.932; # x position of front edge of bottom segment
1198    bot_bx = 71.831; # x position of back edge of bottom segment
1199    top_z = 150.546; # level (z position) of top segment
```

-23-

3164-05_fin.udo 9/12/2007

```
1200    top_fx = 40.386; # x position of front edge of top segment
1201    top_bx = 69.660; # x position of back edge of top segment
1202    f_angle = 28.88; # angle (up from x-axis) of the front edge
1203    b_angle = 138.74; # angle (up from x-axis) of the back edge
1204
1205    if SLICINGPHASE == 0 do # if in draft phase
1206        ELEMENT(z+bot_z, 0, 15, metal, fin_21_b, INn); # bottom element
1207            NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
1208            ADDY(fin_width);
1209            ADDLINE(x+bot_fx, y+fin_w2);
1210            CLOSELINE;
1211        ENDELEM;
1212        ELEMENT(z+top_z, 0, 16, metal, fin_21_c, INn); # cover (top) element
1213            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
1214            ADDY(fin_width);
1215            ADDLINE(x+top_fx, y+fin_w2);
1216            CLOSELINE;
1217        ENDELEM;
1218    endif; # draft phase
1219
1220    if SLICINGPHASE == 1 do # if in final phase
1221        top_index = MESHINDEX(z+top_z+buff);
1222        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
1223        OPENOBJECT(fin_21);
1224            count = 1;
1225            while cur_index < top_index do
1226                bot_level = MESH(cur_index);
1227                top_level = MESH(++cur_index);
1228                if bot_level == (z+bot_z) do
1229                    cur_fx = bot_fx;
1230                    cur_bx = bot_bx;
1231                endif;
1232                ELEMENT(bot_level, 0, 15, metal, fin_21@"_"@count@b, INn); # bottom element
1233                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1234                    ADDY(fin_width);
1235                    ADDLINE(cur_fx, y+fin_w2);
1236                    CLOSELINE;
1237                ENDELEM;
1238                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1239                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1240                ELEMENT(top_level, 0, 16, metal, fin_21@"_"@count@c, INn); # cover (top)
    element
1241                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1242                    ADDY(fin_width);
1243                    ADDLINE(cur_fx, y+fin_w2);
1244                    CLOSELINE;
1245                ENDELEM;
1246                count++;
1247            endwhile;
1248        CLOSEOBJ; # fin_21
1249    endif; # final phase
1250
1251    # fin sector 22
```

-24-

3164-05_fin.udo 9/12/2007

```
1252
1253    # params
1254    bot_z = 150.546; # level (z position) of bottom segment
1255    bot_fx = 40.386; # x position of front edge of bottom segment
1256    bot_bx = 69.660; # x position of back edge of bottom segment
1257    top_z = 151.181; # level (z position) of top segment
1258    top_fx = 42.130; # x position of front edge of top segment
1259    top_bx = 68.936; # x position of back edge of top segment
1260    f_angle = 20.01; # angle (up from x-axis) of the front edge
1261    b_angle = 138.74; # angle (up from x-axis) of the back edge
1262
1263    if SLICINGPHASE == 0 do # if in draft phase
1264        ELEMENT(z+bot_z, 0, 15, metal, fin_22_b, INn); # bottom element
1265            NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
1266            ADDY(fin_width);
1267            ADDLINE(x+bot_fx, y+fin_w2);
1268            CLOSELINE;
1269        ENDELEM;
1270        ELEMENT(z+top_z, 0, 16, metal, fin_22_c, INn); # cover (top) element
1271            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
1272            ADDY(fin_width);
1273            ADDLINE(x+top_fx, y+fin_w2);
1274            CLOSELINE;
1275        ENDELEM;
1276    endif; # draft phase
1277
1278    if SLICINGPHASE == 1 do # if in final phase
1279        top_index = MESHINDEX(z+top_z+buff);
1280        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
1281        OPENOBJECT(fin_22);
1282            count = 1;
1283            while cur_index < top_index do
1284                bot_level = MESH(cur_index);
1285                top_level = MESH(++cur_index);
1286                if bot_level == (z+bot_z) do
1287                    cur_fx = bot_fx;
1288                    cur_bx = bot_bx;
1289                endif;
1290                ELEMENT(bot_level, 0, 15, metal, fin_22@"_"@count@b, INn); # bottom element
1291                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1292                    ADDY(fin_width);
1293                    ADDLINE(cur_fx, y+fin_w2);
1294                    CLOSELINE;
1295                ENDELEM;
1296                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1297                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1298                ELEMENT(top_level, 0, 16, metal, fin_22@"_"@count@c, INn); # cover (top)
    element
1299                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1300                    ADDY(fin_width);
1301                    ADDLINE(cur_fx, y+fin_w2);
1302                    CLOSELINE;
1303                ENDELEM;
```

-25-

3164-05_fin.udo 9/12/2007

```
1304                count++;
1305            endwhile;
1306        CLOSEOBJ; # fin_22
1307    endif; # final phase
1308
1309    # fin sector 23
1310
1311    # params
1312    bot_z = 151.181; # level (z position) of bottom segment
1313    bot_fx = 42.130; # x position of front edge of bottom segment
1314    bot_bx = 68.936; # x position of back edge of bottom segment
1315    top_z = 151.816; # level (z position) of top segment
1316    top_fx = 43.873; # x position of front edge of top segment
1317    top_bx = 63.525; # x position of back edge of top segment
1318    f_angle = 20.01; # angle (up from x-axis) of the front edge
1319    b_angle = 173.31; # angle (up from x-axis) of the back edge
1320
1321    if SLICINGPHASE == 0 do # if in draft phase
1322        ELEMENT(z+bot_z, 0, 15, metal, fin_23_b, INn); # bottom element
1323            NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
1324            ADDY(fin_width);
1325            ADDLINE(x+bot_fx, y+fin_w2);
1326            CLOSELINE;
1327        ENDELEM;
1328        ELEMENT(z+top_z, 0, 16, metal, fin_23_c, INn); # cover (top) element
1329            NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
1330            ADDY(fin_width);
1331            ADDLINE(x+top_fx, y+fin_w2);
1332            CLOSELINE;
1333        ENDELEM;
1334    endif; # draft phase
1335
1336    if SLICINGPHASE == 1 do # if in final phase
1337        top_index = MESHINDEX(z+top_z+buff);
1338        TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
    never happen, because the mesh should already be setup
1339        OPENOBJECT(fin_23);
1340            count = 1;
1341            while cur_index < top_index do
1342                bot_level = MESH(cur_index);
1343                top_level = MESH(++cur_index);
1344                if bot_level == (z+bot_z) do
1345                    cur_fx = bot_fx;
1346                    cur_bx = bot_bx;
1347                endif;
1348                ELEMENT(bot_level, 0, 15, metal, fin_23@"_"@count@b, INn); # bottom element
1349                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1350                    ADDY(fin_width);
1351                    ADDLINE(cur_fx, y+fin_w2);
1352                    CLOSELINE;
1353                ENDELEM;
1354                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1355                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1356                ELEMENT(top_level, 0, 16, metal, fin_23@"_"@count@c, INn); # cover (top)
```

-26-

```
element
                NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
                ADDY(fin_width);
                ADDLINE(cur_fx, y+fin_w2);
                CLOSELINE;
            ENDELEM;
            count++;
        endwhile;
    CLOSEOBJ; # fin_23
endif; # final phase

# fin sector 24

# params
bot_z = 151.816; # level (z position) of bottom segment
bot_fx = 43.873; # x position of front edge of bottom segment
bot_bx = 63.525; # x position of back edge of bottom segment
top_z = 152.451; # level (z position) of top segment
top_fx = 45.617; # x position of front edge of top segment
top_bx = 45.617; # x position of back edge of top segment
f_angle = 20.01; # angle (up from x-axis) of the front edge
b_angle = 177.97; # angle (up from x-axis) of the back edge

if SLICINGPHASE == 0 do # if in draft phase
    ELEMENT(z+bot_z, 0, 15, metal, fin_24_b, INn); # bottom element
        NEWLINE(x+bot_fx, y-fin_w2, x+bot_bx, y-fin_w2);
        ADDY(fin_width);
        ADDLINE(x+bot_fx, y+fin_w2);
        CLOSELINE;
    ENDELEM;
    ELEMENT(z+top_z, 0, 16, metal, fin_24_c, INn); # cover (top) element
        NEWLINE(x+top_fx, y-fin_w2, x+top_bx, y-fin_w2);
        ADDY(fin_width);
        ADDLINE(x+top_fx, y+fin_w2);
        CLOSELINE;
    ENDELEM;
endif; # draft phase

if SLICINGPHASE == 1 do # if in final phase
    top_index = MESHINDEX(z+top_z+buff);
    TEST(top_index > 0, "ERROR: mesh not ready on a for a segment"); # this should
never happen, because the mesh should already be setup
    OPENOBJECT(fin_24);
        count = 1;
        while cur_index < top_index do
            bot_level = MESH(cur_index);
            top_level = MESH(++cur_index);
            if bot_level == (z+bot_z) do
                cur_fx = bot_fx;
                cur_bx = bot_bx;
            endif;
            ELEMENT(bot_level, 0, 15, metal, fin_24@"_"@count@b, INn); # bottom element
                NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
                ADDY(fin_width);
```

3164-05_fin.udo 9/12/2007

```
1409                    ADDLINE(cur_fx, y+fin_w2);
1410                    CLOSELINE;
1411                ENDELEM;
1412                cur_fx = x+bot_fx+((top_level-bot_z)/tan(f_angle));
1413                cur_bx = x+bot_bx+((top_level-bot_z)/tan(b_angle));
1414                ELEMENT(top_level, 0, 16, metal, fin_24@"_"@count@c, INn); # cover (top) element
1415                    NEWLINE(cur_fx, y-fin_w2, cur_bx, y-fin_w2);
1416                    ADDY(fin_width);
1417                    ADDLINE(cur_fx, y+fin_w2);
1418                    CLOSELINE;
1419                ENDELEM;
1420                count++;
1421            endwhile;
1422        CLOSEOBJ; # fin_24
1423    endif; # final phase
1424
1425
1426    MARKFJ(ELEML, ALL, PASSIVE); # mark all fin elements as passive
1427    MARKFJ(OBJECTL, ALL, PASSIVE); # mark all fin objects as passive
1428
1429    # create the bevels
1430    CALL("basic/cyvaob.udo", fin_bevel1, 4, 80, 20, air, 45, 90, "B+", x+(.223), y, z+fin_bot, 12);
1431    CALL("basic/cyvaob.udo", fin_bevel2, 4, 80, 20, air, 270, 315, "B+", x+(.223), y, z+fin_bot, 12);
1432
1433    # cut out the bevel
1434    MARKFJ(OBJECTL, fin_bevel1, ACTIVE);
1435    MARKFJ(OBJECTL, fin_bevel2, ACTIVE);
1436    JOIN(CUT);
1437    DELETE(OBJECTL, fin_bevel1);
1438    DELETE(OBJECTL, fin_bevel2);
1439
1440    MARKFJ(ELEML, ALL, RESET); # clear all markings
1441    MARKFJ(OBJECTL, ALL, RESET); # clear all markings
1442
1443    CLOSEOBJ; #oname
1444
1445    # do rotation
1446    if rot > 0 do
1447        MARK(OBJECT, oname, SET); # mark all elements of oname
1448        ROTATE(360-rot, 0, 0);
1449        MARK(OBJECT, oname, RESET);
1450    endif;
1451
```

sphz.udo

sphz.udo 9/12/2007

```
1   comment="Half sphere sliced with combined (BP) or regular elements ";
2   bitmap="sph0.bmp";
3   # modified version of sp, Gabe Cohn -- 07-13-2007
4
5   PAR("name", oname, sp);
6   PAR("diameter", d,  6);
7   PAR("sectors", n, 16);
8   PAR("medium", med, air);
9   PAR("mesh snapping planes?", limits,"Y");
10  PAR("2 - combined; <0,1> - regular", comb, 2);
11
12  ENDHEADER;
13
14  TEST( n > 2, "n must be greater than 2" );
15  TEST( d > 0, "Diameter must be positive" );
16  #lev2=MESHINDEX(z+d/2);
17  #lev1=MESHINDEX(z-d/2);
18  #TEST((SLICINGPHASE==0)||(lev2-lev1)>1, "mesh not prepared for Final phase" );
19
20  EXPANDMESH(z-d/2,z+d/2);
21
22  OPENOBJECT( oname );
23    r  = d/2; # sphere radius
24    r4 = r/2; # half of the radius -- needed to draw bottom and top of the sphere
25
26    step = 360/n;
27
28    # if needed snap mesh to the bottom and top of the sphere
29    if limits=="Y" do
30      CALL( "elements/specz.udo", oname@"sd", r/10, x, y, z-r, 6 );
31      CALL( "elements/specz.udo", oname@"su", r, x, y, z, 6 );
32    endif;
33
34    # Draft phase
35    if SLICINGPHASE == 0 do
36
37      # 2 elements to draw approx. of lower half
38      ELEMENT( z-r, 0, 15, med, oname@"b2", INn );
39          NEWLINE( x+r4, y, x+r4*cos( step ), y+r4*sin( step ) );
40        edgen = 3;
41        while edgen <= n do
42          ADDLINE( x+r4*cos( ( edgen-1 )*step ), y+r4*sin( ( edgen-1 )*step ) );
43          edgen++;
44        endwhile;
45        CLOSELINE;
46      ENDELEM;
47      ELEMENT( z, 0, 16, med, oname@"c2", INn );
48          NEWLINE( x+r, y, x+r*cos( step ), y+r*sin( step ) );
49        edgen = 3;
50        while edgen <= n do
51          ADDLINE( x+r*cos( ( edgen-1 )*step ), y+r*sin( ( edgen-1 )*step ) );
52          edgen++;
53        endwhile;
54        CLOSELINE;
```

-1-

sphz.udo 9/12/2007

```
55        ENDELEM;
56
57
58      endif;
59
60      # Final phase -- pretty up the sphere
61      if SLICINGPHASE == 1 do
62
63        deltamesh = 0.0001;
64
65        # What are indices of first (starting from the bottom) and last mesh levels?
66        i1 = MESHINDEX( z-(r-deltamesh) );
67        i2 - MESHINDEX( z+(deltamesh) );
68
69
70       if limits=="N" do
71          if MESH(i1)<(z-r)-0.0001 do
72            i1++;
73          endif;
74       endif;
75
76        # Generate consecutive layers of the sphere starting from bottom
77        while i1 < i2 do
78          lev1 = MESH( i1 );
79          lev2 = MESH( i1+1 );
80
81          # Get the heigth of the cells at the sphere's bottom and top
82          deld = ( lev2-lev1 );
83          delu = ( MESH( i2 )-MESH( i2-1 ) );
84
85          # Use them as the width of the lower-most or upper-most layer if the
86          # cyllinder cross-section ends without reaching to the next mesh-level
87            xr1 = deld;
88            xr2 = delu;
89
90          # Find the layer width (its bottom)
91          if abs( lev1-z ) < r do
92            xr1 = sqrt( r*r - ( lev1-z )*( lev1-z ) );
93          endif;
94          # Find the layer width (its top)
95          if abs( lev2-z ) < r do
96            xr2 = sqrt( r*r - ( lev2-z )*( lev2-z ) );
97          endif;
98
99          # Create sphere with combined elements
100         if comb == 2 do
101           # Layer bottom
102           ELEMENT( lev1, 0, 5, med, oname@"b"@i1, INn );
103             NEWLINE( x+xr1, y, x+xr1*cos( step ), y+xr1*sin( step ) );
104             edgen = 3;
105             while edgen <= n do
106               ADDLINE( x+xr1*cos( ( edgen-1 )*step ), y+xr1*sin( ( edgen-1 )*step ) );
107               edgen++;
108             endwhile;
```

-2-

sphz.udo 9/12/2007

```
109             CLOSELINE;
110           ENDELEM;
111           # Layer top
112           ELEMENT( lev2, 0, 6, med, oname@"c"@i1, INn );
113             NEWLINE( x+xr2, y, x+xr2*cos( step ), y+xr2*sin( step ) );
114             edgen = 3;
115             while edgen <= n do
116               ADDLINE( x+xr2*cos( ( edgen-1 )*step ), y+xr2*sin( ( edgen-1 )*step ) );
117               edgen++;
118             endwhile;
119             CLOSELINE;
120           ENDELEM;
121         endif;
122
123         #Create sphere with regular elements
124         if comb <= 1 do
125           if xr1 < xr2 do
126             xr3 = xr1+comb*(xr2-xr1);
127           endif;
128           if xr1>=xr2 do
129             xr3 = xr2+comb*(xr1-xr2);
130           endif;
131
132           ELEMENT( lev1, lev2-lev1, 10, med, oname@"b"@i1, INn );
133             NEWLINE( x+xr3, y, x+xr3*cos( step ), y+xr3*sin( step ) );
134             edgen = 3;
135             while edgen <= n do
136               ADDLINE( x+xr3*cos( ( edgen-1 )*step ), y+xr3*sin( ( edgen-1 )*step ) );
137               edgen++;
138             endwhile;
139             CLOSELINE;
140           ENDELEM;
141         endif;
142         i1++;
143       endwhile;
144     endif;
145   CLOSEOBJ;
146
```

3.4 CST Model

3.4.1 How to build the model

This section will describe in detail how to create the model of the Lindgren 3164-05 described in section 3.1.2. It will also serve as a guide for creating the model of the 3164-05 Inventor files design (section 3.1.1) and the Lindgren 3164-06 (section 3.1.3) which are both variations of the following model.

1. input all variables from Tables 3.1.2 a, b and c.
2. define units
   mm, GHz, ps
3. define background
   Normal, Epsilon=1.0, Mue=1.0
4. define frequency range
   1, 18
5. define boundaries
   Xmin "expanded open"
   Xmax "expanded open"
   Ymin "expanded open"
   Ymax "expanded open"
   Zmin "expanded open"
   Zmax "expanded open"
6. define brick: base:base_bot
   Material “PEC”
   Xrange "-base_w/2", "base_w/2"
   Yrange "-base_w/2", "base_w/2"
   Zrange "base_z-(base_h-cut_depth)", "base_z"
7. define sphere: base:hemi
   Material "Vacuum"
   Axis "y"
   CenterRadius "hemi_r"
   TopRadius "0"
   BottomRadius "0"
   Center "0", "0", "base_z"
   Segments "0"
8. boolean insert shapes: base:base_bot, base:hemi
9. delete shape: base:hemi
10. define brick: base:side_py
    Material "PEC”
    Xrange "-base_w/2", "base_w/2"
    Yrange "cut_w/2", "base_w/2"
    Zrange "base_z", "base_z+cut_depth"
11. transform: base:side_py
    Copy
    Mirror

Center "0", "0", "0"
PlaneNormal "0", "1", "0"
Copy Object Name “base:side_ny”

12. transform: base:side_py
Copy
Rotate
Center "0", "0", "0"
Angle "0", "0", "90"
Copy Object Name “base:side_nx”

13. transform: base:side_nx
Copy
Mirror
Center "0", "0", "0"
PlaneNormal "1", "0", "0"
Copy Object Name “base:side_px”

14. define curve polygon: fin:fin_px
Point "x0", "z0"
LineTo "x1", "z1"
LineTo "x2", "z2"
LineTo "x3", "z3"
LineTo "x4", "z4"
LineTo "x5", "z5"
LineTo "x6", "z6"
LineTo "x7", "z7"
LineTo "x8", "z8"
LineTo "x9", "z9"
LineTo "x10", "z10"
LineTo "x11", "z11"
LineTo "x12", "z12"
LineTo "x13", "z13"
LineTo "x14", "z14"
LineTo "x15", "z15"
LineTo "x16", "z16"
LineTo "x17", "z17"
LineTo "x18", "z18"
LineTo "x19", "z19"
LineTo "x20", "z20"
LineTo "x21", "z21"
LineTo "x22", "z22"
LineTo "x23", "z23"
LineTo "x24", "z24"
LineTo "x25", "z25"
LineTo "x26", "z26"
LineTo "x27", "z27"
LineTo "x28", "z28"

15. transform curve: rotate fin
    Rotate
    Center "0", "0", "0"
    Angle "90", "0", "0"
16. define extrudeprofile: fins:fin_px
    Material "PEC"
    Thickness "fin_thick/2"
17. pick face
    side of fin
18. define extrude: fins:fin2
    Material "PEC"
    Thickness "fin_thick/2"
19. boolean add shapes: fins:fin_px, fins:fin2
20. transform: translate fins:fin_px
    Vector "fin_x_shift", "0", "fin_z_shift"
21. define brick: fins:bevel1
    Material "Vacuum"
    Xrange "-5", ".233"
    Yrange "0", "-10"
    Zrange "-1", "100"
22. transform: fins:bevel1
    Rotate
    Center "0", "0", "0"
    Angle "0", "0", "45"
23. transform: fins:bevel1
    Mirror
    Copy
    Center "0", "0", "0"
    PlaneNormal "0", "1", "0"
    Copy Object Name “fins:bevel1_1”
24. boolean subtract shapes: fins:fin_px, fins:bevel1
25. boolean subtract shapes: fins:fin_px, fins:bevel1_1
26. transform fins:fin_px
    Mirror
    Copy
    Center "0", "0", "0"
    PlaneNormal "1", "0", "0"
    Copy Object Name “fins:fin_nx”
27. transform fins:fin_px
    Rotate
    Copy
    Center "0", "0", "0"
    Angle "0", "0", "90"
    Copy Object Name “fins:fin_py”
28. transform fins:fin_py
    Mirror

Copy
Center "0", "0", "0"
PlaneNormal "0", "1", "0"
Copy Object Name “fins:fin_ny”

29. define cylinder: base:base_top
    Material "PEC"
    OuterRadius "base_top_dia/2"
    InnerRadius "0"
    Axis "z"
    Zrange "base_z+cut_depth", "base_z+cut_depth+base_top_thick"
    Xcenter "0"
    Ycenter "0"
    Segments "0"
30. define brick: base:base_top_cutout
    Material "Vacuum"
    Xrange "-base_top_cutout/2", "base_top_cutout/2"
    Yrange "-base_top_cutout/2", "base_top_cutout/2"
    Zrange "base_z+cut_depth", "base_z+cut_depth+base_top_thick"
31. boolean subtract shapes: base:base_top, base:base_top_cutout
32. define cylinder: shld:shld_base (only if including radiation shield)
    Material "PEC"
    OuterRadius "shld_ir+shld_thick"
    InnerRadius "0"
    Axis "z"
    Zrange "feed_bot-shld_base_thick", "feed_bot"
    Xcenter "0"
    Ycenter "0"
    Segments "0"
33. define cylinder: shld:shld_sides (only if including radiation shield)
    Material "PEC"
    OuterRadius "shld_ir+shld_thick"
    InnerRadius "shld_ir"
    Axis "z"
    Zrange "feed_bot", "feed_top"
    Xcenter "0"
    Ycenter "0"
    Segments "0"
34. define material: AN-73 (only if including absrober)
    Type "Normal"
    Epsilon "5"
    Mue "5"
    Electrical Conductivity "1.2"
    Magnetic Conductivity "1.2"
35. define cylinder: abs:abs_base (only if including absrober)
    Material "AN-73"
    OuterRadius "shld_ir"

InnerRadius "0"
Axis "z"
Zrange "feed_bot", "feed_bot+abs_thick"
Xcenter "0"
Ycenter "0"
Segments "0"

36. boolean insert shapes: abs:abs_base, base:base_bot (only if including absrober)
37. define cylinder: abs:abs_sides (only if including absrober)
    Material "AN-73"
    OuterRadius "shld_ir"
    InnerRadius "shld_ir-abs_thick"
    Axis "z"
    Zrange "feed_bot+abs_thick", "feed_top"
    Xcenter "0"
    Ycenter "0"
    Segments "0"
38. define discrete port: 1
    Type "SParameter"
    Impedance "50.0"
    Voltage "1.0"
    Current "1.0"
    Point1 "0.0", "-gap_w/2", "0.0"
    Point2 "0.0", "gap_w/2", "0.0"
39. define global mesh properties
    a. Feed Only
        Hexahedral
        Lines per wavelength “15”
        Lower mesh limit “15”
        Mesh line ratio limit “10”
        Specials/Refinement “Wavelength refinement”
        Specials/Convert geometry data after meshing
        Specials/use TST cells
        This gives a mesh size of 4,031,720
    b. Feed in Radiation Shield
        Hexahedral
        Lines per wavelength “15”
        Lower mesh limit “15”
        Mesh line ratio limit “10”
        Specials/Refinement “Wavelength refinement”
        Specials/Convert geometry data after meshing
        Specials/use TST cells
        This gives a mesh size of 8,989,376
    c. Feed in Radiation Shield
        Hexahedral
        Lines per wavelength “11”
        Lower mesh limit “10”

Mesh line ratio limit “10”
Specials/Refinement “Low frequency refinement”
Specials/Convert geometry data after meshing
Specials/use TST cells
This gives a mesh size of 16,225,488

4

Simulation Outputs

4.1	S-Parameters	103
4.1.1	Obtaining S-Parameters in QuickWave	103
4.1.2	Obtaining S-Parameters in CST	103
4.2	Far-field Patterns	104
4.2.1	Running Patterns in QuickWave	104
4.2.2	Running Patterns in CST	105
4.2.3	Standard File Format for Patterns	115
4.2.4	Converting QuickWave Output to Standard Format	115
4.2.5	Converting CST Patterns to Standard Format	119
4.2.6	Converting Measured Patterns to Standard Format	123
4.3	E-Fields and H-Fields	123
4.3.1	Viewing and Exporting Fields in QuickWave	123
4.3.2	Viewing and Exporting Fields in CST	123

This section gives a detailed description of how to view and save several types of simulation outputs from both QuickWave and CST MWS. The types the outputs that will be described are S-Parameters, Far-field Patterns, and E and H Fields. In addition, standard file formats and other conventions are discussed.

4.1 S-Parameters

4.1.1 Obtaining S-Parameters in QuickWave

In order to obtain S-parameter outputs from QuickWave, at least one S-parameter port needs to be defined in the structure. This must be done when the structure is defined in QW-Editor (see section 2.1.2). S-parameter postprocessing must also be configured in QW-Editor before the structure is exported to QW-Simulator. This can be done using the Postprocessing dialog, which can be found using Parameters>Postprocessing. Select the [S]-differential check box. If this option is not available, then you must first define an S-parameter port. You can select the frequency range over which to analyze S-parameters as well as the step size. At the bottom of the dialog, there are some advanced options for S-parameter extraction. For more information about these extra options, please consult the manual [4-1].

While the simulation is running in QW-Simulator, S-parameter results can be viewed at anytime during the simulation using View>S Results. You can select which S-parameters are displayed using Setup>View Config (Ctrl+A). To view more outputs, such as port impedances, enter Extended mode using Setup>Switch>Extended Results (Ctrl+E). You can also toggle auto-refresh while the simulation is running using Setup>Switch>Dynamic Draw (D). The axis scale and coordinate system can be changed using Setup>Scale and Setup>Coordinates.

The current view of the S-parameter plot can be saved using Setup>Save Picture. However, it is much more useful to export all S-parameter data into a text file. There are two export formats for S-parameter data. The default format is the QuickWave .da3 file. This format is described in the QW-Simulator manual [4-2]. I highly recommend exporting to “Pure data” format, which has the extension .mat. You can select your desired export formats using Setup>Export Formats. Be aware that you may need to explicitly select your desired export format every time before exporting data, otherwise QuickWave will default to the .da3 format. The data is exported using Setup>Save Results. The software will ask you to name the file with the .da3 extension. This is fine, because it will also create the .mat file if you selected “Pure data.” I use the following name convention for S-parameter data: <projectName>_s.mat.

4.1.2 Obtaining S-Parameters in CST

In order to obtain S-parameter outputs from CST, at least one S-parameter port needs to be defined in the structure. If this is the case, S-parameter outputs will automatically be generated.

S-parameter results can be viewed at anytime while the simulation is running. The results are found in the 1D Results folder in the left panel. There are different folders containing different S-parameter data. You can select which S-parameters are displayed by right-clicking on the S-parameter plot and selecting Select Result Curves. The axis scale and coordinate system can be changed by right-clicking on the S-parameter plot and selecting 1D Plot Properties.

The current view of the S-parameter plot can be saved using File>Export>. However, it is much more useful to export the S-parameter data into a plain text file by selecting File>Export>Plot Data (ASCII). I use the following name convention for S-parameter data: <projectName>_s.txt.

4.2 Far-field Patterns

4.2.1 Running Patterns in QuickWave

In order to obtain far-field patterns in QuickWave, a near-to-far (NTF) box must be defined that completely encloses the structure. An NTF box can be inserted into the structure using the built-in UDO objects found in elib/boxes. See the manual for more details [4-1] [4-3]. NTF postprocessing must also be configured in QW-Editor before the structure is exported to QW-Simulator. This can be done using the Postprocessing dialog, which can be found using Parameters>Postprocessing. Select the Near-to-Far check box. If this option is not available, then you must first define an NTF box. You must select the frequencies to compute far-field patterns using a space delimited list.

Far-field patterns can be computed at any time during the simulation, but they will not be accurate until after the simulator has converged. The simulator is stopped while the antenna patterns are being computed. Standard 2D pattern can be calculated using View>Antenna Results. In the dialog, you must select which variable to sweep (Phi or Theta), the sweep range and step, as well as the value of the variable that it held constant. In addition, you can select the point for which will be the reference for the phase data. This can be changed later, but you can also select the gain reference. After clicking OK, the desired patterns will be calculated. This is a time-consuming process that may take several minutes (depending on the sweep range and resolution). In addition, 3D patterns can be obtained using View>Antenna 3D Results. However, this document only discusses 2D patterns. See the manual for more details on 3D patterns [4-2].

Once the plot is displayed, you can change the gain reference using Setup>Antenna Settings>Gain Reference. For all of my saved data, I used the Directivity gain reference. However, selecting Fields at 1 m may be more useful, because that it what I use for CST. By default, only a single frequency from the NTF frequency list will be plotted. You can change the plotted frequency using Setup>Antenna Settings>NTF Freq. or by pressing F. Alternately, you can select which patterns are displayed using Setup>View Config (Ctrl+A). The axis scale and coordinate system can be changed using Setup>Scale and Setup>Coordinates.

The current view of the far-field pattern plot can be saved using Setup>Save Picture. However, it is much more useful to export all far-field data into a text file. There are two export formats for far-field patterns. The default format is the QuickWave .da3 file. This format is described in the QW-Simulator manual [4-2]. I highly recommend exporting to “Pure data” format, which has the extension .mat. You can select your desired export formats using Setup>Export Formats. Be aware that you may need to explicitly select your desired export format every time before exporting data, otherwise QuickWave will default to the .da3 format. The data is exported using Setup>Save Results. The software will ask you to name the file with the .da3 extension. This is fine, because it will also create the .mat file if you selected “Pure data.” I use the following naming convention for far-field pattern data: <projectName>_ant_p<phi>.mat. For example, the following four filenames will contain the patterns of the project named ‘test’ at four different slices, phi = 0,45,90,135: test_ant_p000.mat, test_ant_p045.mat, test_ant_p090.mat, test_ant_p135.mat.

The format of the .mat "Pure data" output from QuickWave is grouped by frequency. For each frequency, there is a one line header and then the data follows immediately after. The header contains 4 values: <Frequency> <RadiatedPower> <RadiationEfficieny> <RadiationResistance>. The data has 5 columns: <Angle> <mag(Etheta)> <phase(Etheta)> <mag(Ephi)> <phase(Ephi)>. Section 4.2.4 describes how this file can be converted into a file that can be easily input into Matlab for postprocessing.

Clearly, exporting far-field patterns for a large number of slices can be very time consuming and quite cumbersome for the user. For this reason, there is a way of automating this process. It involves defining breakpoints before running the simulator. Breakpoints will perform some specified action at a given iteration number in the simulation. To create a breakpoint, open the Breakpoint editor using Configure>Breakpoints while the simulator is stopped or suspended. Click add, and then name the breakpoint, select the iteration number, and the action to perform from the list on the left. To create far-field patterns select Save Antenna Results Extended. Then type the desired filename for the output file. Click Export Formats to select the desired output formats. Click Antenna Settings to select the options for the given far-field pattern. Click OK, and then define more breakpoints. From experience, it appears that each breakpoint must have a unique iteration number. For this reason, I usually set up each slice to be computed at the final iteration number plus the phi value for the given slice. For example, if I want the following slices: phi = 0, 45, 90, 135, and the simulation will have converted after 24000 iterations then I will setup the following breakpoints:

Breakpoint	Iteration
freeze	23999
ant_p000	24000
ant_p045	24045
ant_p090	24090
ant_p135	24135
suspend	24180

As a warning, the breakpoints are saved between runs of the simulator, but the export format is not saved. As a result, before running a previously saved simulation you will need to manually go through each of the breakpoints and redefine the desired export formats (all other settings are saved however). If you do not do this, then QuickWave will use the default .da3 format.

4.2.2 Running Patterns in CST

In order to obtain far-field patterns in CST, farfield monitors need to be setup for each desired frequency. A single farfield monitor can be added using Solve>Field Monitors, and then selecting Farfield/RCS, and the desired frequency. In order to define several farfield monitors over a wide frequency with a constant frequency step, a built-in macro can be used. The macro can be found using Macros>Farfield>Broadband Farfield Monitors. This will allow defining the frequency range and frequency step size. This macro will automatically insert all of the desired farfield monitors. In order to run far-field patterns, the boundary conditions must be open space, perhaps with the exception of a ground plane.

Far-field patterns must be computed after the entire simulation has completed. The results are found in the Farfields folder in the left panel. There is a different folder for each farfield monitor. By clicking on the folder, the far-field patterns will be computed. By default, they are plotted in 3D mode. You can change the default settings by right-clicking on the far-field plot and selecting Plot Properties.

In the Plot Properties dialog, all of the pattern settings can be selected. On the General tab, the plot type can be selected. For common Cartesian 2D plots, select Cartesian. Then select which angle to vary, and which slice to use for the constant angle. At the bottom, there is an option for displaying 360-degree patterns rather than 180-degree patterns, and for plotting from -180 to 180 rather than 0 to 360. From the Plot Mode tab, you select the gain reference and scaling. For all of my saved data, I have used E-field at 1 m reference. The Gain (IEEE) output is a more useful quantity; however, I am interesting the phase of the E-fields as well as the Gain. In addition, I do not use the farfield approximation. From the Origin tab, you can select the reference point for the phase data. I always use the origin of the coordinate system, which is the feed point of the antenna. From the Phase Center tab, you can set CST to automatically calculate the phase center from the pattern. I have not used this; however, it may be very useful. After clicking OK, the patterns will be computed for the selected settings. This is time-consuming process that may take several minutes (depending on the settings).

Once the plot is displayed, you can change the axis scale by right-clicking on the plot and selecting 1D Plot Properties. In addition, making certain changes from the Plot Properties dialog will simply change the plot without re-computing the pattern.

The current view of the plot can be saved using File>Export>BMP. However, it is much more useful to export the far-field data into a plain text file. This can be done using File>Export>Plot Data (ASCII). I use the following name convention for far-field pattern data: <projectName>_ant_p<phi>_f<freq>.txt. For example, the following table shows some example filenames for the project named test.

Phi	Freq	Filename
0	2.2	test_ant_p000_f2.2.txt
90	2.2	test_ant_p090_f2.2.txt
45	4.0	test_ant_p045_f4.txt
135	4.0	test_ant_p135_f4.txt

The format of the ASCII output from CST contains more data than I usually care about. As a result, I have script that will convert it to the standard format for far-field pattern data, which can be easily imported into Matlab. This script is discussed in section 4.2.5.

When you click on the folder for another farfield monitor, it will compute the pattern using the same settings as the last one. Therefore, to obtain patterns for a large number of frequencies, you have to click on each monitor and then export the data by hand. In addition, for obtaining data from multiple slices, this whole process must be repeated. Clearly, it is not feasible to do this manually for many monitors or many slices. As a result, I wrote two macros to automate this process. Each macro is described below. Both macros can be found in the Farfield directory.

Broadband Farfield Save:

This macro will automatically compute and export patterns over a given frequency range, with a constant frequency step size. In addition, it will compute several slices for each frequency given a constant phi step size. This macro assumes that the farfield monitors were created using Broadband Farfield Monitors macro. When you select this macro, a dialog will prompt you for the following settings:

fname_base: base name of the files containing all of the pattern outputs. By convention this should be the project name
- Type: string
- Default: <projectName>

freq_low: lower frequency limit to compute patterns
- Type: floating-point
- Default: <fmin>

freq_high: upper frequency limit to compute patterns
- Type: floating-point
- Default: <fmax>

freq_step: frequency step for computing patterns
- Type: floating-point
- Default: 0.5

phi_step: phi step for computing patterns at a given frequency
- Type: integer
- Default: 90

This macro operates based on the following simple pseudo-code:

for `freq` between `freq_low` and `freq_high` in steps of `freq_step`:
  for `phi` between 0 and 179 in steps of `phi_step`:
    get pattern for `phi` slice at `freq` frequency
    export pattern using standard file naming convention

All the patterns are computed using the following settings:

PlotType: Cartesian
Vary: Theta
ThetaStep: 1 deg
ThetaRange: [-180, 180]
UseFarfieldApproximation: No
PlotMode: E-field
RefDistance: 1 m
Units: dB V/m
Origin: (0,0,0)
PolVector: (0,1,0)
Coords: Spherical
PhaseCenter: No

Output file naming convention: <`fname_base`>_ant_p<`phi`>_f<`freq`>.txt

Specified Freq. Farfield Save:

This macro will automatically compute and export patterns for a list of specified frequencies. In addition, it will compute several slices for each frequency given a constant phi step size. The farfield monitor names must be either the default naming (CST auto-labeling), or the name created by the Broadband Farfield Monitors macro. When you select this macro, a dialog will prompt you for the following settings:

fname_base: base name of the files containing all of the pattern outputs. By convention this should be the project name
- Type: string
- Default: <projectName>

freq list: space delimited list of frequencies to evaluate
- Type: string: space delimited list of floating-points
- Default: <fmin>

phi_step: phi step for computing patterns at a given frequency
- Type: integer
- Default: 90

This macro operates based on the following simple pseudo-code:

```
for freq in freq_list:
    for phi between 0 and 179 in steps of phi_step:
        get pattern for phi slice at freq frequency
        export pattern using standard file naming convention
```

All the patterns are computed using the same settings as Broadband Farfield Save. In addition, the same output file naming convention is used.

As a warning, the farfield monitors corresponding to requested frequencies must exit. If the macro attempts to compute the patterns for a farfield monitor name that does not actually exist, then no error will be displayed. The last selected monitor will be used again but saved as though it were the current frequency. This can cause invalid data in the output files, which is completely unknown to the user. Therefore, be sure that the specified frequencies exist and follow the correct naming convention.

The source code for both macros is included on the following pages.

Broadband Farfield Save – page 109

Specified Freq. Farfield Save – page 112

Broadband Farfield Save Macro

Broadband Farfield Save^+MWS.mcs 9/17/2007

```
' Farfield / Broadband Farfield Save
' !!! Do not change the line above !!!
'-----------------------------------------------------------------------------
' History Of Changes
'-----------------------------------------------------------------------------
' 09-05-2007 - Gabe Cohn        - Macro Created
'-----------------------------------------------------------------------------
Option Explicit

Sub Main ()

With FarfieldPlot
    .Reset
    .Plottype ("cartesian")
    .Vary ("theta")
    .Step (1)
    .UseFarfieldApproximation (False)
    .SetTheta360 (True)
    .CartSymRange (True)
    .DrawIsoLongitudeLatitudeLines (False)

    .SetPlotMode ("Efield")
    .Distance (1)
    .SetScaleLinear (False)
    .SetLogRange (40.0)
    .DBUnit ("0")
    .EnableFixPlotMaximum (False)

    .Origin ("zero")
    .AlignToMainLobe (False)
    .PolarizationVector (0.0, 1.0, 0.0)
    .SetCoordinateSystemType ("spherical")

    .UseDecouplingPlane (False)
    .SetUserDecouplingPlane (False)
    .UseMirrorPlane (False)
    .SetUserMirrorPlane (False)
    .MirrorPlaneAxis ("X")
    .MirrorPlanePositicn (0.0)
    .EnablePhaseCenterCalculation (False)
End With

With ASCIIExport
    .Reset
    .Mode ("FixedWidth")
End With

'' Modified from Farfield/Broadband Farfield Monitors 09-05-2007
Dim cst_fflow  As Double
Dim cst_ffhigh As Double
Dim cst_ffstep As Double
Dim cst_ffphistep As Double

Dim cst_action As String
```

-1-

```
55  Dim cst_fffname As String
56
57  BeginHide
58
59      Begin Dialog UserDialog 300,266,"Farfield Monitors to Process and Save",.DialogFunc
60          Text 20,14,90,14,"fname_base",.Text1
61          Text 20,70,90,14,"freq_low",.Text2
62          Text 20,98,90,14,"freq_high",.Text3
63          Text 20,126,90,14,"freq_step",.Text4
64          Text 20,182,90,14,"phi_step",.Text5
65          TextBox 120,7,160,21,.fname
66          TextBox 120,63,160,21,.flow
67          TextBox 120,91,160,21,.fhigh
68          TextBox 120,119,160,21,.fstep
69          TextBox 120,175,160,21,.phistep
70          OKButton 40,238,90,21
71          CancelButton 170,238,90,21
72      End Dialog
73  Dim dlg As UserDialog
74
75  ' Get Initial Values
76  cst_fffname = GetProjectBaseName
77  With Solver
78      cst_fflow = .GetFmin
79      cst_ffhigh= .GetFmax
80  End With
81  ' Store Initial Values
82  dlg.fname   = cst_fffname
83  dlg.flow    = CStr(cst_fflow)
84  dlg.fhigh   = CStr(cst_ffhigh)
85  dlg.fstep   = "0.5"
86  dlg.phistep = "90"
87
88  If (Dialog(dlg)=0) Then
89      cst_action="cancel"
90  Else
91      cst_action="create"
92  End If
93
94
95  If cst_action = "create" Then
96
97      cst_fffname     = dlg.fname
98      cst_fflow       = Evaluate(dlg.flow)
99      cst_ffhigh      = Evaluate(dlg.fhigh)
100     cst_ffstep      = Evaluate(dlg.fstep)
101     cst_ffphistep   = Evaluate(dlg.phistep)
102
103     assign "cst_fffname"
104     assign "cst_fflow"
105     assign "cst_ffhigh"
106     assign "cst_ffstep"
107     assign "cst_ffphistep"
108 End If
```

Broadband Farfield Save^+MWS.mcs **9/17/2007**

```
109
110   assign "cst_action"
111
112   ScreenUpdating False
113
114   EndHide
115
116   If cst_action = "create" Then
117
118       Dim cst_ffrq As Double
119       Dim phi As Integer
120       Dim freq As Double
121
122       ' safety against rounding errors for upper limit (tiny increase of cst_ffhigh)
123       cst_ffhigh = cst_ffhigh + 0.000001 * cst_ffstep
124
125       For cst_ffrq=cst_fflow To cst_ffhigh STEP cst_ffstep
126           SelectTreeItem("Farfields\ff_"+Replace(Format(cst_ffrq,"00.0000"),",",".")+" ↵
      [1]")
127           For phi = 0 To 179 STEP cst_ffphistep
128               FarfieldPlot.Phi(phi)
129               FarfieldPlot.Plot
130               freq = cst_ffrq
131               If freq - Int(freq) = 0 Then
132                   freq = Int(freq)
133               End If
134               ASCIIExport.FileName(cst_fffname+"_ant_p"+Format(phi,"000")+"_f"+CStr(freq ↵
      )+".txt")
135               ASCIIExport.Execute
136           Next phi
137       Next cst_ffrq
138
139       ScreenUpdating True
140   End If
141
142   End Sub
143
144   '' Copied from Farfield/Broadband Farfield Monitors 09-05-2007
145   Function DialogFunc%(Item As String, Action As Integer, Value As Integer)
146       Select Case Action
147       Case 1 ' Dialog box initialization
148       Case 2 ' Value changing or button pressed
149           Select Case Item
150           Case "Help"
151               StartHelp "common_preloadedmacro_Farfield_Broadband_Farfield_Monitors"
152               DialogFunc = True
153           End Select
154       Case 3 ' ComboBox or TextBox Value changed
155       Case 4 ' Focus changed
156       Case 5 ' Idle
157       End Select
158   End Function
159
```

Specified Freq. Farfield Save Macro

Specified Freq. Farfield Save^+MWS.mcs 9/17/2007

```
' Farfield / Specified Freq. Farfield Save
' !!! Do not change the line above !!!
'-----------------------------------------------------------------------------
' History Of Changes
' This Macro Is a slight modification of Farfield/Broadband Farfield Save
'-----------------------------------------------------------------------------
' 09-09-2007 - Gabe Cohn        - Macro Created
'-----------------------------------------------------------------------------
Option Explicit

Sub Main ()

With FarfieldPlot
    .Reset
    .Plottype ("cartesian")
    .Vary ("theta")
    .Step (1)
    .UseFarfieldApproximation (False)
    .SetTheta360 (True)
    .CartSymRange (True)
    .DrawIsoLongitudeLatitudeLines (False)

    .SetPlotMode ("Efield")
    .Distance (1)
    .SetScaleLinear (False)
    .SetLogRange (40.0)
    .DBUnit ("0")
    .EnableFixPlotMaximum (False)

    .Origin ("zero")
    .AlignToMainLobe (False)
    .PolarizationVector (0.0, 1.0, 0.0)
    .SetCoordinateSystemType ("spherical")

    .UseDecouplingPlane (False)
    .SetUserDecouplingPlane (False)
    .UseMirrorPlane (False)
    .SetUserMirrorPlane (False)
    .MirrorPlaneAxis ("X")
    .MirrorPlanePositicn (0.0)
    .EnablePhaseCenterCalculation (False)
End With

With ASCIIExport
    .Reset
    .Mode ("FixedWidth")
End With

'' Modified from Farfield/Broadband Farfield Monitors 09-05-2007
Dim cst_ffphistep As Double

Dim cst_action As String
Dim cst_fffreqs As String
Dim cst_fffname As String
```

-1-

Specified Freq. Farfield Save^+MWS.mcs 9/17/2007

```
55
56  BeginHide
57
58      Begin Dialog UserDialog 390,210,"Farfield Monitors to Process and Save",.DialogFunc
59          Text 20,14,90,14,"fname_base",.Text1
60          Text 20,70,90,14,"freq list",.Text2
61          Text 20,126,90,14,"phi_step",.Text3
62          TextBox 120,7,250,21,.fname
63          TextBox 120,63,250,21,.freqs
64          TextBox 120,119,250,21,.phistep
65          OKButton 70,182,90,21
66          CancelButton 230,182,90,21
67      End Dialog
68  Dim dlg As UserDialog
69
70  ' Get Initial Values
71  cst_fffname = GetProjectBaseName
72  With Solver
73      cst_fffreqs = CStr(.GetFmin)
74  End With
75  ' Store Initial Values
76  dlg.fname   = cst_fffname
77  dlg.freqs   = cst_fffreqs
78  dlg.phistep = "90"
79
80  If (Dialog(dlg)=0) Then
81      cst_action="cancel"
82  Else
83      cst_action="create"
84  End If
85
86
87  If cst_action = "create" Then
88
89      cst_fffname     = dlg.fname
90      cst_fffreqs     = dlg.freqs
91      cst_ffphistep   = Evaluate(dlg.phistep)
92
93      assign "cst_fffname"
94      assign "cst_fffreqs"
95      assign "cst_ffphistep"
96  End If
97
98  assign "cst_action"
99
100 ScreenUpdating False
101
102 EndHide
103
104 If cst_action = "create" Then
105
106     Dim cst_ffrq As Double
107     Dim begi As Integer
108     Dim endi As Integer
```

-2-

Specified Freq. Farfield Save^+MWS.mcs **9/17/2007**

```
109       Dim phi As Integer
110       Dim freq As Double
111
112       begi = 1
113       endi = InStr(begi+1, cst_fffreqs, Space(1))
114       If endi = 0 Then endi = Len(cst_fffreqs)+1
115
116       While begi <= Len(cst_fffreqs)
117           cst_ffrq = Evaluate(Mid(cst_fffreqs, begi, endi-begi))
118           SelectTreeItem("Farfields\farfield (f="+Replace(Format(cst_ffrq,"#.0"),",",".")+") [1]")
119           SelectTreeItem("Farfields\ff_"+Replace(Format(cst_ffrq,"00.0000"),",",".")+" [1]")
120           For phi = 0 To 179 STEP cst_ffphistep
121               FarfieldPlot.Phi(phi)
122               FarfieldPlot.Plot
123               freq = cst_ffrq
124               If freq - Int(freq) = 0 Then
125                   freq = Int(freq)
126               End If
127               ASCIIExport.FileName(cst_fffname+"_ant_p"+Format(phi,"000")+"_f"+CStr(freq)+".txt")
128               ASCIIExport.Execute
129           Next phi
130           begi = endi+1
131           endi = InStr(begi+1,cst_fffreqs, Space(1))
132           If endi = 0 Then endi = Len(cst_fffreqs)+1
133       Wend
134
135       ScreenUpdating True
136   End If
137
138   End Sub
139
140   '' Copied from Farfield/Broadband Farfield Monitors 09-05-2007
141   Function DialogFunc%(Item As String, Action As Integer, Value As Integer)
142       Select Case Action
143       Case 1 ' Dialog box initialization
144       Case 2 ' Value changing or button pressed
145           Select Case Item
146           Case "Help"
147               StartHelp "common_preloadedmacro_Farfield_Broadband_Farfield_Monitors"
148               DialogFunc = True
149           End Select
150       Case 3 ' ComboBox or TextBox Value changed
151       Case 4 ' Focus changed
152       Case 5 ' Idle
153       End Select
154   End Function
155
```

4.2.3 Standard File Format for Patterns

There is a standard file format for far-field patterns. This is required in order for the postprocessing functions described in section 5 to correctly import data into Matlab. The remainder of section 4.2 describes how to convert the far-field outputs of QuickWave and CST to the standard file format The naming convention of the standard file format is: <projectName>_ant_p<phi>_f<freq>.mat. For example, the following table shows some example filenames for the project named 'test'.

Phi	Freq	Filename
0	2.2	test_ant_p000_f2.2.mat
90	2.2	test_ant_p090_f2.2.mat
45	4.0	test_ant_p045_f4.mat
135	4.0	test_ant_p135_f4.mat

The format of the standard file is five whitespace delimited columns:

Column 1: theta angle in degrees
Column 2: magnitude of E_theta on a linear scale (V/m)
Column 3: phase of E_theta in degrees
Column 4: magnitude of E_phi on a linear scale (V/m)
Column 5: phase of E_phi in degrees

The theta angles should be integers in the range of -180 to 180. Although most (and maybe all) of the postprocessing function may work fine if this is not the case, they have not been tested and therefore may not work properly.

In addition, by convention the polarization vector is along the y-axis. This means that phi=0 is the H-Plane, and phi=90 is the E-Plane. Notice that this is the opposite definition used for all measured data.

4.2.4 Converting QuickWave Output to Standard Format

The QuickWave export format is very similar to the standard format, so conversion is very simple. The QuickWave format is actually the same, except all frequencies are found in a single file, and they also contain a header line. The script to convert from QuickWave to the standard format basically just separates out each frequency into separate files.

This script is written in python and is called sepData.py. It is intended to be run from the command prompt on any computer that has the python interpreter installed. The command line interface is:

sepData.py <infile> [<outfile_base>] [-p <start_phi:phi_step:end_phi>]

Arguments and Options:

infile: Required. Name of input file. If running only one file, then this should be the full filename. If running multiple files, this should be the base of the filenames including the _ant, but not the _p###.

outfile_base:	Optional. Base name of output file. If running only one file, then this should be the full filename. If running multiple files, this should be the base of the filenames including the _ant, but not the _p###. If this is omitted, then the same base as the input filename will be used.
-p option:	Optional. Allows running more than one file, by converting several different phi slices for a given project. Must define the phi range and step size. The specification is in the Matlab array format, where both endpoints will be evaluated.

Sample calls on the project named ‘test’:

To convert a single file:
sepData.py test_ant_p000.mat or sepData.py test_ant_p000
To convert a single file using a different output file basename:
sepData.py test_ant_p000 newName_ant_p000
To convert all slices with a phi resolution of 15 deg:
sepData.py test_ant -p 0:15:179
To convert only the principal planes:
sepData.py test_ant -p 0:90:179

The following pages contain the source code for sepData.py.

sepData.py 9/18/2007

```
#!/usr/bin/python
# script to separate out all of the different frequencies of QuickWave data so that they can be processed by Matlab
# Gabe Cohn, 07-20-07

import sys

def main():
    '''runs from command prompt'''

    usage = "sepData <infile> [<outfile_base>] [-p <start_phi:step:end_phi>]"

    args = sys.argv[1:]
    phis = [None]
    if '-p' in args:
        try:
            i = args.index('-p')
            args.pop(i)
            phi_rng = args.pop(i).split(':')
        except:
            print "must specify an argument for -p option"
            print usage
            return
        if len(phi_rng) != 3:
            print "error in -p argument"
            print usage
            return
        phis = range(int(phi_rng[0]), int(phi_rng[2])+1, int(phi_rng[1]))

    if len(args) > 2 or len(args) < 1:
        print "\nERROR: this script requires 1 or 2 arguments, %i given" %len(args)
        print usage
        return

    inname = args[0]
    outname = None
    if len(args) == 2:
       outname = args[1]
    for phi in phis:
        phi_str = ''
        if phi != None:
            phi_str = '_p%03d' %phi
        if outname == None:
            split(inname+phi_str)
        else:
            split(inname+phi_str, outname)
    return

def split(infile, outfile_base=None):
    '''splits a QuickWave .mat file into several .mat files that can be imported into MATLAB'''
    if outfile_base == None:
        outfile_base = infile.rstrip('.mat') + '_f'

```

-1-

sepData.py 9/18/2007

```
53      try:
54          inf = file(infile, 'r')
55      except:
56          try:
57              inf = file(infile+'.mat', 'r')
58          except:
59              print '\n', infile, "not found."
60              return
61
62      i = 0
63      line = inf.readline()
64      while line:
65          i += 1
66          elems = line.split()
67          if len(elems) == 4: # start of next data set (frequency set)
68              # get simple string representation of frequency
69              freq = float(elems[0])
70              if freq - int(freq):
71                  freq = str(freq)
72              else:
73                  freq = str(int(freq))
74              try: # close last file
75                  outf.close()
76              except UnbcundLocalError:
77                  pass
78              try: # open next file
79                  outf = file(outfile_base+freq+'.mat', 'w')
80              except:
81                  print '\n', "can't create", outfile
82                  return
83          else:
84              # copy whole line to the outfile
85              outf.write(line)
86          line = inf.readline() # get next line
87
88      outf.close()
89      inf.close()
90      return
91
92  if __name__ == '__main__':
93      main()
```

4.2.5 Converting CST Output to Standard Format

Converting the CST export format to the standard format is more complicated. First, only certain columns of the CST format are needed. In addition, the values need to be converted from dB to a linear scale. The matrix of data also needs to be reshaped in order to get the theta values in order from -180 to 180.

This script is written in python and is called cst2mat.py. It is intended to be run from the command prompt on any computer that has the python interpreter installed. The command line interface is:

cst2mat.py <infile>[*]

Arguments and Options:

infile: Required. Name of input file. If running only one file, then this should be the full filename. If running multiple files, this should be the base of the filenames, and an asterisk should be appended to the end. The program will attempt to convert all files that start with <infile> and have the extension '.txt'. If a file does not appear to fit the CST export format, then a warning message is displayed and the file is ignored. All output files will have the same name as the input file, but the extension will be changed from '.txt' to '.mat'.

Sample calls on the project named 'test':

To convert a single file:

cst2mat.py test_ant_p000_f2.2.txt

To convert all slices:

cst2mat.py test_ant*

In order to perform the matrix operations required for the conversion, this script requires the external module numarray. I used numarray version 1.3.3; however, other versions should work as well. For more information about numarray, see the website [4-4].

The following pages contain the source code for cst2mat.py.

cst2mat.py 9/18/2007

```
#!/usr/bin/python
# script to convert the farfield output files from CST into files that can be imported easily into Matlab (in the standard format)
# Gabe Cohn, 08-29-2007

import sys, os
try:
    from numarray import array, zeros, shape
    from numarray import concatenate as cat
except ImportError:
    print 'external module numarray not found; please install numarray'
    raise

def main():
    '''runs from command prompt'''

    usage = "cst2mat <infile>[*]"

    if len(sys.argv[1:]) != 1:
        print "\nERROR: this script takes only 1 arguments, %i given" %len(args)
        print usage
        return

    inname = sys.argv[1]
    if inname.endswith('*'):
        inname = inname[:-1]
        allfiles = os.listdir('./')
        files = []
        for file in allfiles:
            if file.startswith(inname):
                files.append(file)
        for file in files:
            convert(file)
    else:
        convert(inname)

    return

def convert(infile):
    '''converts a CST .txt file into a .mat file that can be imported into MATLAB'''

    if not infile.endswith('.txt'): # only accept '.txt'
        return

    try:
        inf = file(infile, 'r')
    except:
        try:
            inf = file(infile+'.txt', 'r')
        except:
            print '\n', infile, "not found."
            return

    outfile = infile.rstrip('.txt')
```

cst2mat.py 9/18/2007

```
54
55      # load in each line, save only important values, while converting to linear units
56      # then reshape matrix to be theta: [-180, 180]
57      # then save to file
58
59      for i in range(3): # skip the first two lines, and get the third line ready
60          line = inf.readline()
61      # format of line:  Theta [deg.]    Phi [deg.]    Abs(EAbs)[dBV/m ] Abs(ETh)[dBV/m ]  Phase(ETh)[deg.]  Abs(EPh)[dBV/m ]  Phase(EPh)[deg.]     Left [dBV/m ]    Right [dBV/m ]
62
63      try:
64          phi = line.split()[1]
65      except: # won't work if it is the wrong type of file
66          print "invalid file: wrong number of lines,  \'%s\'" %infile
67          inf.close()
68          return
69
70      i_change = None # index of the phi change (representing start of negative theta values)
71
72      data = zeros((0,5), 'Float32')
73      # format: [ theta [deg], |E_theta| [lin], <E_theta [deg], |E_phi| [lin], <E_phi [deg] ]
74
75      while line:
76          line = line.split()
77          if i_change == None and line[1] != phi:
78              i_change = shape(data)[0]
79
80          if len(line) != 9: # check line format
81              print "invalid file: wrong number of lines,  \'%s\'" %infile
82              inf.close()
83              return
84
85          try:
86              newdata = zeros((1,5), 'Float32')
87              newdata[0][0] = float(line[0])                      # theta     [deg]
88              if i_change:
89                  newdata[0][0] *= -1 # make angle negative
90              newdata[0][1] = 10**(float(line[3])/20)             # |E_theta| [lin]
91              newdata[0][2] = float(line[4])                      # <E_theta  [deg]
92              if newdata[0][2] > 180:
93                  newdata[0][2] -= 360 # put angle in [-180,180]
94              newdata[0][3] = 10**(float(line[5])/20)             # |E_phi|   [lin]
95              newdata[0][4] = float(line[6])                      # <E_phi    [deg]
96              if newdata[0][4] > 180:
97                  newdata[0][4] -= 360 # put angle in [-180,180]
98              data = cat((data, newdata))
99          except:
100             print "invalid file: has string data, \'%s\'" %infile
101             inf.close()
102             return
103
```

-2-

cst2mat.py 9/18/2007

```
104             line = inf.readline() # get next line
105
106         inf.close()
107         assert i_change != None # must have found a change in phi
108
109         newdata = zeros((1, 5), 'Float32') # matrix used to reshape the data matrix
110         newdata[0] = data[i_change-1] # theta = 180 (also -180)
111         newdata[0][0] *= -1 # make first line theta = -180
112         newdata = cat((newdata, data[i_change:]))
113         newdata = cat((newdata, data[:i_change]))
114         assert shape(newdata)[0] == shape(data)[0]+1 # check size of matrix
115
116         try: # open next file
117             outf = file(outfile+'.mat', 'w')
118         except:
119             print '\n', "can't create", outfile
120             return
121
122         for i in range(shape(newdata)[0]):
123             outf.write('%16.6E  %16.7E %16.7E %16.7E %16.7E\n' %tuple((newdata[i])))
124
125         outf.close()
126         return
127
128     if __name__ == '__main__':
129         main()
```

4.2.6 Converting Measured Patterns to Standard Format

The measured patterns are stored in excel files. I currently do not have an automated method for converting this data into the standard file format for far-field patterns. I have been creating excel worksheets, which have the data in the correct 5 column format. Then I have been saving these worksheets in tab delimited plain text format, with the standard naming convention. This is the best way that we currently have for converting measured pattern to the standard format.

Be sure that the standard format files have the magnitude values in linear units; not in dB. All measured data is in dB, so it needs to be changed before saving to the standard format.

In addition, the measured data and standard format use a different coordinate system. The measured data defines the E-Plane to phi=0, where the standard format uses the convention that the E-Plane is phi=90. To eliminate confusion, I have been labeling all files in standard format using the convention that phi=0 is the H-Plane and phi=90 is the E-Plane.

4.3 E-Fields and H-Fields

4.3.1 Viewing and Exporting Fields in QuickWave

There are no requirements for the structure or postprocessing options in order to view fields in QuickWave. Fields can be viewed at anytime during the simulation using View>Fields. You can toggle auto-refresh while the simulation is running using Setup>Switch>Dynamic Draw (D). There are many view modes for viewing fields. All of the options can be selected using the Setup menu. For more information, please consult the manual [4-2].

The current view can be saved using Setup>Save Picture. However, it is much more useful to export all field data into a text file. There are several ways of doing this. All of these can be found in the setup menu. The options are Dump Fields, Save Envelope, and Save Volume Env./Inst. I have not exported field data using QuickWave. For information about exporting field, refer to the manual [4-2].

4.3.2 Viewing and Exporting Fields in CST

In order to obtain E and H field patterns in CST, field monitors need to setup for each desired frequency. A field monitors can be added using Solve>Field Monitors, and then selecting either E-Field or H-Field/Surface current, and the desired frequency. Field monitors can be defined over the entire volume of the structure or at set cut-planes.

The results are found in the 2D/3D Results folder in the left panel. There are separate folders for E-Fields and H-Fields. Inside these folders, there is a different folder for each field monitor. By clicking on the folder, the field patterns will be computed and displayed. The view can be changed using the normal controls. The fields will be automatically recalculated as the view changes.

The current view can be saved using File>Export>BMP. However, it is much more useful to export the field data into a plain text file. This can be done using File>Export>Plot Data (ASCII).

For more information about field monitors, refer to the manual [4-5].

5

External Postprocessing

5.1	Convert Pattern Data to Co- and Cross-Polar Form	126
5.1.1	coxpol.m	126
5.2	Plotting Far-field Patterns	131
5.2.1	plot_pattern.m	131
5.2.2	auto_patterns.m	136
5.2.3	auto_cmp_patterns.m	139
5.3	Phase Center Analysis	142
5.3.1	phase_center.m	142
5.3.2	change_ref.m	146
5.3.3	auto_pc.m	151
5.4	Computation of Radiated Power, Taper and Spillover Efficiencies	155
5.4.1	eff.m	155

This section discusses all of the Matlab scripts that were created to perform external postprocessing of the far-field data. These functions perform tasks such as data conversion, data plotting, phase center calculation, and efficiency calculation. Most of the results presented in section 6 were prepared using these scripts.

All postprocessing scripts were created and used in Matlab 7.4.0 (R2007a).

5.1 Convert Pattern Data to Co- and Cross-Polar Form

The output format of far-field patterns from both QuickWave and CST is in spherical coordinates, where the gain is expressed in terms of E_θ and E_ϕ. However, it is often more useful to have the gain expressed in terms of a co-polarization and a cross-polarization.

5.1.1 coxpol.m

The Matlab script coxpol.m computes the co-polar and cross-polar patterns from Ephi and Etheta data for files in the standard file format for far-field patterns. The script will do the computation for several frequencies, and several different phi slices.

Usage: coxpol(fname_base, phis, freqs, [thetas])

Arguments:

fname_base:	base filename does not include the _p#, _f#, or .mat
phis:	list of slice planes (phi values) to evaluate. Import filenames are derived from this
freqs:	list of frequencies to evaluate; Import filenames are derived from this
thetas:	(optional) list of theta values to evaluate. Allows you to use a different theta_step, for example: -180:10:180 defaults to -180:180

File Outputs:

for all files, the angles and phases are in deg, the frequencies are in GHz, and the magnitudes are in dB

<fname_base>_p<phi>_coPol_mag.txt - contains magnitude of co-polar pattern versus theta for each frequency for the given phi

<fname_base>_p<phi>_coPol_phi.txt' - contains phase of co-polar pattern versus theta for each frequency for the given phi

<fname_base>_p<phi>_xPol_mag.txt' - contains magnitude of cross-polar pattern versus theta for each frequency for the given phi

<fname_base>_p<phi>_xPol_phi.txt' - contains phase of cross-polar pattern versus theta for each frequency for the given phi

Sample calls on the project named ‘test’:

To get files containing the frequencies 2.2, 4, 8.4, 11, 14, and 18 for the slices phi=0,45,90,135:

```
coxpol(test_ant, 0:45:179, [2.2 4 8.4 11 14 18])
```

To get files containing all frequencies between 1 and 18 with a 0.5 GHz step for only the principal planes, and only computing thetas with a theta_step of 10 degrees:

```
coxpol(test_ant, [0 90], 1:0.5:18, -180:10:180)
```

This script operates based on the following pseudo-code:

```
for phi in phis:
    for freq in freqs:
        load data from file for given phi and freq
    convert to co-polar and cross-polar
    write output files
```

The conversion from spherical E_θ and E_ϕ to co-polar (E_p) and cross-polar (E_q) fields is defined on page 28 of Rudge [5-1]. Assuming that the polarization vector is along the y-axis, as it is for data in the standard file format, then the conversion is:

$$E_p(\theta,\phi) = E_\theta(\theta,\phi)\sin\phi\ \mathbf{a}_\theta + E_\phi(\theta,\phi)\cos\phi\ \mathbf{a}_\phi$$

$$E_q(\theta,\phi) = E_\theta(\theta,\phi)\cos\phi\ \mathbf{a}_\theta - E_\phi(\theta,\phi)\sin\phi\ \mathbf{a}_\phi$$

The following pages contain the source code for coxpol.m.

coxpol.m 9/19/2007

```
function coxpol(varargin)
% computes the co-polar and cross-polar patterns from Ephi and Etheta for
% several frequencies
%
% usage: coxpol(fname_base, phis, freqs, [thetas])
%
% INPUTS:
% fname_base: the base filename, does not include the _p#, _f#, or .mat
% phis: list of planes (phi values) to evaluate; filenames are derived from
%       this
% freqs: list of frequencies to evaluate; filenames are derived from this
% thetas: (optional) list of theta values to evaluate
%   defaults to -180:180
% OUTPUTS:
% creates the following new files:
%   for all files, the angles and phases are in deg, the frequnecies are in
%   GHz, and the magnitudes are in dB
%   fname_base+'_p'+phi+'_coPol_mag.txt' - contains magnitude of co-polar
%       pattern versus theta for each frequency for the given phi
%   fname_base+'_p'+phi+'_coPol_phi.txt' - contains phase of co-polar
%       pattern versus theta for each frequency for the given phi
%   fname_base+'_p'+phi+'_xPol_mag.txt' - contains magnitude of cross-polar
%       pattern versus theta for each frequency for the given phi
%   fname_base+'_p'+phi+'_xPol_phi.txt' - contains phase of cross-polar
%       pattern versus theta for each frequency for the given phi
%
% File Name:    coxpol.m
% Author:       Gabe Cchn
%               Caltech, RF and Microwave Group
% Dates:        08-21-2007

assert(nargin >= 3 && nargin <= 4, 'ERROR: this function takes between 3 and 4 a
arguments');

% default values
thetas = -180:180;

fname_base = varargin{1};
phis = varargin{2};
freqs = varargin{3};
if nargin == 4
    thetas = varargin{4};
end

% check args
assert(ischar(fname_base), 'ERROR: the first argument (fname_base) must be a string');
assert(isvector(phis), 'ERROR: the second argument (phis) must be a vector');
assert(~isempty(phis), 'ERROR: the second argumnet (phis) is empty');
assert(isvector(freqs), 'ERROR: the third argument (freqs) must be a vector');
assert(~isempty(freqs), 'ERROR: the third argumnet (freqs) is empty');
assert(isvector(freqs), 'ERROR: the fourth argument (thetas) must be a vector');
assert(~isempty(freqs), 'ERROR: the fourth argument (thetas) is empty');

files = dir([fname_base,'*']);
```

-1-

coxpol.m 9/19/2007

```
54  assert(~isempty(files), ['ERROR: no files found that match ', fname_base]);
55  disp('Created the follcwing files:');
56
57  for phi = phis
58
59      theta_ind = 0;
60      E_theta = size(length(freqs), length(thetas));
61      E_phi = size(length(freqs), length(thetas));
62      for f_ind = 1:length(freqs)
63          f_string = num2str(freqs(f_ind));
64          filename = [fname_base,'_p',sprintf('%03d',phi),'_f',f_string,'.mat'];
65          found = 0;
66          for i = 1:length(files)
67              if strmatch(files(i).name, filename, 'exact')
68                  found = 1;
69              end
70          end
71          if found == 0
72              disp(['file ', filename, ' not found']);
73              continue
74          end
75          data = load('-ascii', filename); % load data set
76
77          if theta_ind == 0
78              theta_ind = zeros(size(thetas));
79              for i = 1:length(thetas)
80                  ind = find(round(data(:,1)) == thetas(i),1);
81                  assert(~isempty(ind), 'ERROR: theta value not found in data set');
82                  theta_ind(i) = ind;
83              end
84          end
85
86          for i = 1:length(theta_ind)
87              assert(thetas(i) == round(data(theta_ind(i),1)), 'ERROR: inconsistant data');
88              E_theta(f_ind, i) = data(theta_ind(i),2)*exp(complex(1,pi/180*data(
    theta_ind(i),3)));
89              E_phi(f_ind, i) = data(theta_ind(i),4)*exp(complex(1,pi/180*data(theta_ind(
    i),5)));
90          end
91
92          clear data; % clear data from memory
93      end
94
95      % get data
96      co_mag = 20*log10(abs(E_theta*sind(phi) + E_phi*cosd(phi))); % in dB
97      co_phase = 180/pi*angle(E_theta*sind(phi) + E_phi*cosd(phi)); % in deg
98      x_mag = 20*log10(abs(E_theta*cosd(phi) - E_phi*sind(phi))); % in dB
99      x_phase = 180/pi*angle(E_theta*cosd(phi) - E_phi*sind(phi)); % in deg
100
101     % create output files
102     writeFile(freqs, thetas, co_mag, [fname_base,'_p',sprintf('%03d',phi),
    '_coPol_mag.txt']);
103     disp(sprintf('\t%s', [fname_base,'_p',sprintf('%03d',phi),'_coPol_mag.txt']));
104     writeFile(freqs, thetas, co_phase, [fname_base,'_p',sprintf('%03d',phi),
```

-2-

coxpol.m 9/19/2007

```
'_coPol_phase.txt']);
105       disp(sprintf('\t%s', [fname_base,'_p',sprintf('%03d',phi),'_coPol_phase.txt']));
106       writeFile(freqs, thetas, x_mag, [fname_base,'_p',sprintf('%03d',phi),
'_xPol_mag.txt']);
107       disp(sprintf('\t%s', [fname_base,'_p',sprintf('%03d',phi),'_xPol_mag.txt']));
108       writeFile(freqs, thetas, x_phase, [fname_base,'_p',sprintf('%03d',phi),
'_xPol_phase.txt']);
109       disp(sprintf('\t%s\n', [fname_base,'_p',sprintf('%03d',phi),'_xPol_phase.txt']));
110    end
111    return
112    end
113
114    function writeFile(freqs, thetas, data, fname)
115    % creates output file
116    outfile = fopen(fname, 'w');
117    fprintf(outfile, 'Freq (GHz)\t');
118    for i = 1:length(thetas)
119        fprintf(outfile, '%d\t', round(thetas(i)));
120    end
121    fprintf(outfile, '\n');
122    for i = 1:length(freqs)
123        fprintf(outfile, '%f\t', freqs(i));
124        for j = 1:length(thetas)
125           fprintf(outfile, '%f\t', data(i,j));
126        end
127        fprintf(outfile, '\n');
128    end
129    fclose(outfile);
130    end
131
```

5.2 Plotting Far-Field Patterns

The following scripts create presentation quality plots of far-field patterns. The script plot_pattern.m is the basic function that creates the plots. auto_patterns.m is a script to quickly create plots of several frequencies of the same slice all plotted on one axis. auto_cmp_patterns.m is a script to quickly create plots for comparing patterns from different files.

5.2.1 plot_pattern.m

The Matlab script plot_pattern.m is the low-level function for creating plots of far-field patterns. The plots created are considered to be of presentation quality, and can be shrunk down without losing important details. This function can create a new figure object, or it can simply add a plot to an active axis.

Usage: h = plot_pattern(thetas, data, type, [name], [LineSpec], [ColorSpec], [title], [h])

Arguments:

thetas:	vector of x-values, which are theta values in degrees
data:	vector of data to be converted to y-values. This is |E| in linear units
type:	'abs' or 'rel' if 'abs': will plot data in dB {20*log10(data)} if 'rel': will plot data relative to the value at theta=0 {20*log10(data/data(theta=0))}
name:	(optional) name to display on legend (can omit with [] or '')
LineSpec:	(optional) LineSpec string to define how line is to be displayed (can omit with [] or '') defaults to 'r' (creates a red line)
ColorSpec:	(optional) ColorSpec array to define the color of the line, this will override any color defined using LineSpec (can omit with [])
title:	(optional) title to display above axes (can omit with [] or '')
h:	(optional) handle of axis object to add plot to if h is specified, then all plot properties will be preserved if h is not defined, a new figure will be opened, and the default style will be used (font size, axes labels, axes ranges)

Screen Output:

if h input is specified, then a plot will be added to the axes handle h
if h input in not specified, then a new figure will be created, and the data will be plotted

Return:

h: the axes handle of plot

Sample calls:

To create a new plot of absolute data:
h = plot_pattern(thetas, data, ‘abs’)
To create a new plot of relative data while labeling the data on the legend as ‘Relative’:
h = plot_ pattern (thetas, data, ‘rel’, ‘Relative’)
To create a new plot of a dashed green line of relative data with no legend label:
h = plot_ pattern (thetas, data, ‘rel’, ‘’, ‘--g’)
To create a new plot of an orange line of relative data with no legend label and an axis title of “Test Data”:
h = plot_ pattern (thetas, data, ‘rel’, ‘’, ‘’, [1 .6 0], ‘Test Data’)
To add to an existing axis with a plot of an orange line of relative data with the legend ‘Test’, and the two-line axis title of “More”, “Data”:
h = plot_ pattern (thetas, data, ‘rel’, ‘Test’, ‘’, [1 .6 0], {‘More’;‘Data’}, h)

The following pages contain the source code for plot_pattern.m.

plot_pattern.m 9/19/2007

```
function h = plot_pattern(varargin)
% plots far-field radiation patterns
%
% usage: plot_pattern(thetas, data, type, [name], [LineSpec], [ColorSpec], [title], [h])
%
% INPUTS:
% thetas: vector of x-values, which are theta values in degress
% data: vector of data to be converted to y-values.  This is |E| in linear
%   units
% type: 'abs' or 'rel'
%   if 'abs': will plot data in dB [20*log10(data)]
%   if 'rel': will plot data relative to the value at theta=0
%             [20*log10(data/data(theta-0))]
% name: (optional) name to display on legend (can omit with [] or '')
% LineSpec: (optional) LineSpec string to define how line is to be
%   displayed (can omit with [] or '')
%   defaults to 'r'
% ColorSpec: (optional) ColorSpec array to define the color of the line,
%   this will override any color defined using LineSpec (can omit with [])
% title: (optional) title to display above axes (can omit with [] or '')
% h: (optional) handle of axis object to add plot to
%   if h is specified, then all plot properties will be preserved
%   if h is not defined, a new figure will be opened, and the default style
%       will be used (font size, axes labels, axes ranges)
% OUTPUTS:
% if h input is specified, then a plot will be added to the axes handle h
% if h input in not specified, then I new figure will be created, and the
%   data will be plotted
% the handle h of the axes will be returned.
%
% File Name:    plot_pattern.m
% Author:       Gabe Cchn
%               Caltech, RF and Microwave Group
% Dates:        08-18-2007
%
% Revisions:    08-29-2007, Gabe Cohn - added ColorSpec option
%               09-16-2007, Gabe Cohn - added interpolation
%                                     - changed default grid style
%                                     - changed default line thickness

assert(nargin >= 3 && nargin <= 8, 'ERROR: this function takes bewteen 3 and 8 arguments');

% set defaults for args
name = ''; % no name
LineSpec = 'r'; % default to red line
ColorSpec= []; % use default color
plot_title = ''; % no title
h = []; % no handle

% get args
if nargin >= 8
    h = varargin{8};
end
```

-1-

plot_pattern.m 9/19/2007

```
54  if nargin >= 7
55      plot_title = varargin{7};
56  end
57  if nargin >= 6
58      ColorSpec = varargin{6};
59  end
60  if nargin >= 5 && ~strcmpi(varargin{5},'')
61      LineSpec = varargin{5};
62  end
63  if nargin >= 4
64      name = varargin{4};
65  end
66  type = varargin{3};
67  data = varargin{2};
68  thetas = varargin{1};
69
70  % check args
71  assert(isvector(thetas), 'ERROR: the first argument (thetas) must be a vector');
72  assert(isvector(data), 'ERROR: the second argument (data) must be a vector');
73  assert(length(thetas) == length(data), 'ERROR: the first and second arguments (thetas
    data) do not have the same length');
74  assert(~isempty(data), 'ERROR: the first and second arguments (thetas and data) are
    empty');
75  assert(ischar(type), 'ERROR: the third argument (type) must be either ''abs'' or
    ''rel''');
76  if ~strcmpi(type,'abs') && ~strcmpi(type,'rel')
77      disp('WARNING: type is not ''abs'' or ''rel'' it will be assumed to be ''abs''');
78      type = 'abs';
79  end
80  assert(ischar(LineSpec), 'ERROR: LineSpec must be a string');
81
82  if min(size(h) == [0 0]) % if no handle specified
83      h = createNewAxis(type);
84  end
85  if ~ishold
86      hold(h);
87  end
88
89  if strcmpi(type,'rel')
90      i = find(thetas == 0);
91      assert(~isempty(i), 'ERROR: Data set does not include theta=0, can do ''rel''');
92      data = data./data(i);
93  end
94  data = 20*log10(data); % put in dB
95  % interpolate to get resolution of 1 deg if not already
96  if length(thetas) ~= length(-180:180)
97      data = interp1(thetas,data,-180:180,'cubic');
98      thetas = -180:180;
99  end
100
101 line_h = plot(thetas, data, LineSpec, 'LineWidth', 4);
102 if min(size(ColorSpec) ~= [0 0])
103     set(line_h, 'Color', ColorSpec);
104 end
```

-2-

plot_pattern.m 9/19/2007

```
105   if min(size(name) ~= [0 0])
106       set(line_h, 'DisplayName', name);
107       legend('Location','Best');
108   end
109   if min(size(plot_title) ~= [0 0])
110       title(plot_title, 'FontSize', 42);
111   end
112
113   hold off;
114   return
115   end
116
117
118   function h = createNewAxis(type)
119   fullscreen = get(0, 'ScreenSize');
120   figure('Position', [1 0.03*fullscreen(4) fullscreen(3) 0.9*fullscreen(4)], 'Color',
      'white');
121   h = axes('Box', 'on');
122   hold(h);
123   set(h, 'XLim', [-180 180], 'XTick', -180:45:180, 'XMinorTick', 'on');
124   set(h, 'YLim', [-35 5], 'YTick', -35:5:5);
125   set(h, 'GridLineStyle', '-');
126   set(h, 'LineWidth', 4);
127   grid;
128   if strcmpi(type, 'abs')
129       set(h, 'YLimMode', 'auto', 'YTickMode', 'auto');
130   end
131   set(h, 'FontSize',30);
132   xlabel('Elevation Angle (theta) [deg]');
133   ylabel('dB');
134   end
```

5.2.2 auto_patterns.m

The Matlab script auto_patterns.m automatically plots radiation patterns for a single slice at several different frequencies

Usage: auto_patterns(fname_base, freqs, [title], [pol], [type])

Arguments:

fname_base:	the base filename, does not include the _f#, or .mat
freqs:	list of frequencies to plot; filenames are derived from this
title:	(optional) title of plot
pol:	(optional) which polarization to plot, Etheta or Ephi must be either 'phi' or 'theta' defaults to 'phi'
type:	(optional) 'abs' or 'rel' for absolute or relative defaults to 'rel'

Screen Output:

creates a new figure window with of the plots on a single axis

Sample calls for the project name ‘test’:

To create a plot of the relative Ephi data for the frequencies 2.2, 4, 8.4, 11, 14, and 18:

auto_patterns(‘test_ant_p000’, [2.2 4 8.4 11 14 18])

To create a plot of the absolute Etheta data for the frequencies 2.2, 4, 8.4, 11, 14, and 18:

auto_patterns(‘test_ant_p090’, [2.2 4 8.4 11 14 18], ‘’, ‘theta’, ‘abs’)

To create a plot of the relative Ephi data for the frequencies 2.2, 4, 8.4, 11, 14, and 18, and name title it with the two line title “Test Data”, “Selected Freqs”:

auto_patterns(‘test_ant_p000’, [2.2 4 8.4 11 14 18], {‘Test Data’;‘Selected Freqs’}, ‘phi’, ‘rel’)

The following future improvements could be made to this script. Currently, the only plot “polarizations” allowed are Etheta (‘theta’) and Ephi (‘phi’). It would useful to add the following polarization types:

‘abs’	-	plots the absolute magnitude of the E-field sqrt(|Etheta|^2+|Ephi|^2)
‘cpol’	-	plots the co-polar E-field using transformation in coxpol.m
‘xpol’	-	plots the cross-polar E-field using transformation in coxpol.m

The following pages contain the source code for auto_patterns.m.

auto_pattems.m 9/19/2007

```
function auto_patterns(varargin)
% automatically plots radiation patterns for a single slice at several
% frequencies
%
% usage: auto_patterns(fname_base, freqs, [title], [pol], [type])
%
% INPUTS:
% fname_base: the base filename, does not include the _f#, or .mat
% freqs: list of frequencies to plot; filenames are derived from this
% title: (optional) title of plot
% pol: (optional) which polarization to plot, Etheta or Ephi
%   must to a string, either 'phi' or 'theta'
%   defaults to 'phi'
% type: (optional) 'abs' or 'rel' for absolute or relative
%   defaults to 'rel'
% OUTPUTS:
% creates a new figure window with of the plots
%
% File Name:    auto_patterns.m
% Author:       Gabe Cohn
%               Caltech, RF and Microwave Group
% Dates:        08-18-2007
%
% Revisions:    08-29-2007, Gabe Cohn - changed default colors

assert(nargin >= 2 && nargin <= 5, 'ERROR: this function takes between 2 and 5 arguments');

% default values
pol = 4; % 'phi'
type = 'rel';
title = ''; % no title

fname_base = varargin{1};
freqs = varargin{2};
if nargin >= 3
    title = varargin{3};
end
if nargin >= 4
    assert(ischar(varargin{4}), 'fourth arg must be either ''phi'' or ''theta''');
    if strcmpi(varargin{4},'theta')
        pol = 2;
    end % else, default to 'phi' (4)
end
if nargin >= 5
    type = varargin{5};
end

% check args
assert(ischar(fname_base), 'ERROR: the first argument (fname_base) must be a string');
assert(isvector(freqs), 'ERROR: the second argument (freqs) must be a vector');
assert(~isempty(freqs), 'ERROR: the second argumnet (freqs) is empty');
assert(ischar(type), 'ERROR: the fifth argument (type) must be either ''abs'' or ''rel''');
```

-1-

auto_pattems.m 9/19/2007

```
53  if ~strcmpi(type,'abs') && ~strcmpi(type,'rel')
54      disp('WARNING: type is not ''abs'' or ''rel'' it will be assumed to be ''rel''');
55      type = 'rel';
56  end
57  files = dir([fname_base,'*']);
58  assert(~isempty(files), ['ERROR: no files found that match ', fname_base]);
59
60  % select line colors
61  default_colors = [ 255  0   0;      % red
62                     255  153 0;      % orange
63                     255  255 0;      % yellow
64                     51   153 0;      % green
65                     51   51  255;    % blue
66                     153  102 255;    % violet
67                     ]/255;
68  colors = zeros([length(freqs) 3]);
69  i_end = min(size(default_colors,1), size(colors,1));
70  colors(1:i_end,:) = default_colors(1:i_end,:);
71
72  h = []; % start value of handle
73  for f_i = 1:length(freqs)
74      f_string = num2str(freqs(f_i));
75
76      filename = [fname_base,'_f',f_string,'.mat'];
77      found = 0;
78      for i = 1:length(files)
79          if strmatch(files(i).name, filename, 'exact')
80              found = 1;
81          end
82      end
83      if found == 0
84          disp(['file ', filename, ' not found']);
85          continue
86      end
87      data = load('-ascii', filename); % load data set
88
89      h = plot_pattern(data(:,1), data(:,pol), type, [f_string, ' GHz'], '', colors(f_i
    ,:), title, h);
90
91      clear data; % clear data from memory
92  end
93  return
94  end
95
```

-2-

5.2.3 auto_cmp_patterns.m

The Matlab script auto_cmp_patterns.m automatically plots radiation patterns from different files as a comparison. This can be very useful for comparing different data sets at a given frequency and slice. For example, comparing measured data to simulation results.

Usage: auto_cmp_patterns(fnames, [pol], [type], [title])

Arguments:

names:	semi-colon delimited list of full filenames
pol:	(optional) which polarization to plot, Etheta or Ephi must be either 'phi' or 'theta' defaults to 'phi'
type:	(optional) 'abs' or 'rel' for absolute or relative defaults to 'rel'
title:	(optional) title of plot

Screen Output:

creates a new figure window with of the plots on a single axis

Sample calls:

To create a plot comparing the relative Ephi data at phi=0 for 4 GHz between the project ‘test’, ‘test2’ and ‘test3’:

```
auto_cmp_patterns(‘test_ant_p000_f4.mat ; test2_ant_p000_f4.mat ;
    test3_ant_p000_f4.mat’)
```

To create a plot comparing the absolute Etheta data at phi=90 for 8.4 GHz between the project ‘test’, ‘test2’ and ‘test3’:

```
auto_cmp_patterns(‘test_ant_p090_f8.4.mat ; test2_ant_p090_f8.4.mat ;
    test3_ant_p090_f8.4.mat’, ‘theta’, ‘abs’)
```

To create a plot comparing the absolute Etheta data at phi=90 for 8.4 GHz between the project ‘test’, ‘test2’ and ‘test3’ with the title “Test 1 2 3 Comparison”:

```
auto_cmp_patterns(‘test_ant_p090_f8.4.mat ; test2_ant_p090_f8.4.mat ;
    test3_ant_p090_f8.4.mat’, ‘theta’, ‘abs’, ‘Test 1 2 3 Comparison’)
```

The following future improvements could be made to this script. Currently, the only plot “polarizations” allowed are Etheta (‘theta’) and Ephi (‘phi’). It would useful to add the following polarization types:

‘abs’	-	plots the absolute magnitude of the E-field sqrt(|Etheta|^2+|Ephi|^2)
‘cpol’	-	plots the co-polar E-field using transformation in coxpol.m
‘xpol’	-	plots the cross-polar E-field using transformation in coxpol.m

The following pages contain the source code for auto_cmp_patterns.m.

auto_cmp_patterns.m 9/19/2007

```
function auto_cmp_patterns(varargin)
% automatically plots radiation patterns from different files
% useful for comparing different data sets at a given frequency and slice
%
% usage: auto_cmp_patterns(fnames, [pol], [type], [title])
%
% INPUTS:
% fnames: semi-colon delimited list of full filenames
% pol: (optional) which polarization to plot, Etheta or Ephi
%   must to a string, either 'phi' or 'theta'
%   defaults to 'phi'
% type: (optional) 'abs' or 'rel' for absolute or relative
%   defaults to 'rel'
% title: (optional) title of plot
% OUTPUTS:
% creates a new figure window with of the plots
%
% File Name:    auto_cmp_patterns.m
% Author:       Gabe Cohn
%               Caltech, RF and Microwave Group
% Dates:        08-29-2007
%
% Revisions:    09-19-2007, Gabe Cohn - changed order of colors

assert(nargin >= 1 && nargin <= 4, 'ERROR: this function takes between 1 and 4 arguments');

% default values
pol = 4; % 'phi'
type = 'rel';
title = ''; % no title

fnames = varargin{1};
if nargin >= 2
    assert(ischar(varargin{2}), 'second arg must be either ''phi'' or ''theta''');
    if strcmpi(varargin{2},'theta')
        pol = 2;
    end % else, default to 'phi' (4)
end
if nargin >= 3
    type = varargin{3};
end
if nargin >= 4
    title = varargin{4};
end

% check args
assert(ischar(fnames), 'ERROR: the first argument (fnames) must be a string');
assert(ischar(type), 'ERROR: the third argument (type) must be either ''abs'' or ''rel''');
if ~strcmpi(type,'abs') && ~strcmpi(type,'rel')
    disp('WARNING: type is not ''abs'' or ''rel'' it will be assumed to be ''rel''');
    type = 'rel';
end
```

-1-

auto_cmp_patterns.m 9/19/2007

```
53
54  % select line colors
55  default_colors = [ 255  0   0;      % red
56                     51   51  255;    % blue
57                     255  153 0;      % orange
58                     51   153 0;      % green
59                     153  102 255;    % violet
60                     255  255 0;      % yellow
61                     ]/255;
62  colors = zeros([length(find(fnames==';'))+1, 3]);
63  i_end = min(size(default_colors,1), size(colors,1));
64  colors(1:i_end,:) = default_colors(1:i_end,:);
65  num = size(colors,1);
66
67  h = []; % start value of handle
68  while ~isempty(fnames)
69      num = num - 1;
70      i = find(fnames == ';',1); % get index of next ';'
71      if ~isempty(i)
72          fname = strtrim(fnames(1:i-1));
73          fnames = strtrim(fnames(i+1:length(fnames)));
74      else
75          fname = strtrim(fnames);
76          fnames = '';
77      end
78
79      files = dir(fname);
80      if isempty(files)
81          disp(['file ', fname, ' not found']);
82          continue
83      end
84
85      data = load('-ascii', fname); % load data set
86
87      h = plot_pattern(data(:,1), data(:,pol), type, fname, '', colors(size(colors,1)-num
    ,:), title, h);
88
89      clear data; % clear data from memory
90  end
91  return
92  end
93
```

5.3 Phase Center Analysis

The following scripts are used to manipulate the phase data from the far-fields. Most importantly, these scripts allow the computation of the phase center of the feed. The script phase_center.m is the basic function that computes the phase center for a given frequency. change_ref.m will phase shift far-field data in order to represent a different reference plane. auto_pc.m is a script to quickly compute the phase center variation over several different frequencies.

5.3.1 phase_center.m

The Matlab script phase_center.m computes the optimum phase center of an antenna based on the far-field pattern. The optimization is done using the phase data from several planes (phi slices)

Usage: [z_pc, dev] = phase_center(angles, phases, freq, [angle_range])

Arguments:

angles:	vector of angle data in degrees (x-vals, usually theta)
phases:	matrix of phase data (degrees) corresponding to each angle each row represents a data set for a single plane (phi slice)
freq:	measurement frequency (GHz)
angle_range:	[angle_min, angle_max] angle range to evaluate phase over all angles are in degrees this allows the user to restrict the computation to the main beam defaults to [-45.1,44.9] this gives [-45,45] after decimal error

Returns:

z_pc:	the position (on the z-axis) of the phase center (in m)
dev:	the standard deviation of the phases calculated from z_pc (in deg)

This script operates based on the following pseudo-code:

```
unwrap phases
removed phase discontinuity in CST MWS data
add a large multiple of 360 to phases to put it in the far field
restrict phase data to main beam (defined by angle_range)
optimize z_pc
        vary the reference point (z_pc) until the phase variation of the far-field data is
                minimized over all slices
return z_pc, dev (phase deviation at z_pc)
```

See section 5.3.2 for information about the change of reference that is done in the optimization step.

The following pages contain the source code for phase_center.m.

phase_center.m 9/19/2007

```
function [z_pc, dev] = phase_center(varargin)
% finds the phase center of an antenna based on the far-field pattern
% find the best phase over several planes
%
% usage: phase_center(angles, phases, freq, [angle_range])
%
% INPUTS:
% angles: angle data in degrees (x-vals, usually theta) {vector}
% phases: phase data (degrees) corresponding to each angle in angles
%   each row represents a data set for a single plane
%   {matrix}
% freq: measurement frequency (GHz)
% angle_range: [angle_min, angle_max] angle range to evaluate phase over
%   {2-dim row vector}
%   all angles are in degrees
%   this allows the user to restrict the computation to the main beam
%   defaults to [-45.1,44.9] this gives [-45,45] after decimal error
% OUTPUTS:
% z_pc: the position (on the z-axis) of the phase center (in m)
% dev:  the standard deviation of the phases calculated from z_pc (in deg)
%
% File Name:    phase_center.m
% Author:       Gabe Cohn
%               Caltech, RF and Microwave Group
% Dates:        07-19-2007; 08-08-2007
% Revisions:    09-10-2007 - Gabe Cohn      - added CST phase shift

% set arg defaults
angle_range = [-45.1,44.9];

assert(nargin >= 3 && nargin <= 4, 'ERROR: this function takes 3 or 4 arguments');

if nargin == 4
    angle_range = varargin{4};
end
angles = varargin{1};
phases = varargin{2};
freq = varargin{3};

% check args
assert(isvector(angles), 'ERROR: the first argument (angles) must be a vector');
assert(isnumeric(phases), 'ERROR: the second argument (phases) must be numeric');
assert(isscalar(freq), 'ERROR: the third argument (freq) must be a scalar');
assert(isvector(angle_range) && length(angle_range) == 2, 'ERROR: the fourth argument (angle_range) must be a 2-vector');

if isvector(phases) && size(phases, 1) ~= 1 % if phases is a column vector
    phases = phases'; % make it a row vector
end
assert(length(angles) == size(phases, 2), 'ERROR: the first two arguments (angles, phases) must have the same number of data points');

% find indicies corresponding the angle_range
angle_index = [0,length(angles)]; % default to using entire range
```

-1-

phase_center.m 9/19/2007

```
53  i = 1;
54  while (i <= length(angles)) && (angles(i) < angle_range(1))
55      i = i + 1;
56  end
57  if i > length(angles)
58      disp('ERROR: all angles are less than the min range value');
59      disp('will use the entire data set');
60  else
61      angle_index(1) = i;
62      while (i <= length(angles)) && (angles(i) <= angle_range(2))
63          i = i + 1;
64      end
65      if i < angle_index(2)
66          angle_index(2) = i;
67      end
68  end
69
70  i = 1;
71  while angles(i) ~= 0
72      i = i + 1;
73  end
74  assert(i <= size(phases, 2), 'ERROR: data does not include 0 deg');
75
76  phases = 180/pi*unwrap(pi/180*phases, [], 2); % unwrap the phases (in degrees)
77
78  % CST always puts a phase discontinuity between theta=-1 and theta=0
79  % I don't know why this is, but this code should remove that discontinuity
80  mid = find(angles == 0, 1);
81  for i = 1:size(phases,1)
82      if abs(round(phases(i,mid-1)-phases(i,mid))) == 180
83         phases(i,1:mid-1) = phases(i,1:mid-1)-round(phases(i,mid-1)-phases(i,mid));
84      end
85  end
86
87  phases = phases + 1e5*360; % put them all in the far field
88      % 1e5 is just a big number, the greater this number the better the fit,
89      % at the expense of computation time
90
91  % get rid of all data cutside of valid range
92  angles = angles(angle_index(1):angle_index(2));
93  phases = phases(:,angle_index(1):angle_index(2));
94
95  done = 0;
96  lims = [0, .15]; % initial guess on the limits for the phase center (in m)
97  while done == 0
98      [z_pc, dev] = fminbnd(@(z_pc) phaseDev(z_pc, angles, phases, freq), lims(1), lims(2
    ));
99      if z_pc < lims(1)+.01 || z_pc > lims(2)-.01 % if best z_pc is close to the lim
100         lims = lims + [-0.05, 0.05]; % expand lims
101     else % value is good, we are done
102         done = 1;
103     end
104 end
105
```

-2-

phase_center.m 9/19/2007

```
106   %disp(['z_pc = ', num2str(z_pc), ' with std = ', num2str(dev)]);
107   return
108
109   function dev = phaseDev(z_pc, thetas, phases, freq)
110   % returns the standard deviation in phase computed from the reference of
111   % z_pc (guess of phase center)
112   % z_pc is in meters
113   % thetas and phases are in degrees
114   % freq is the measurement frequency in GHz
115   z_pc_phase = 360*z_pc*freq*1e9/2.9979e8; % get z_pc in phase (deg)
116   devs = zeros(1, size(phases, 1));
117   for i = 1:length(devs)
118       equiPhase = zeros(size(thetas));
119       for j = 1:length(thetas)
120           equiPhase(j) = getEquiPhase(z_pc_phase, thetas(j), phases(i,j));
121       end
122       devs(i) = std(equiPhase);
123   end
124   dev = mean(devs);
125   return
126
127   function equiPhase = getEquiPhase(z_pc_phase, theta, phase)
128   % given (theta, phase) from the origin, it returns the phase to the same
129   % point from a new reference (guessed phased center) that is on the
130   % vertical axis, but displaced from the origin by z_pc_phase (in degrees)
131   % all angles are in degrees
132   edge_pt = phase * [sind(theta), cosd(theta)];
133   equiPhase = norm(edge_pt - [0, z_pc_phase]); % distance (in phase) between
134                                  % the edge_pt and the guessed phase center
135   return
136
```

5.3.2 change_ref.m

The Matlab script change_ref.m takes a far-field data file and phase shifts all of the phase data to represent a new reference point (phase center). A separate reference will be used for each frequency, and several planes (phi slices) can be evaluated at each frequency. The value of the reference is the displacement from the original reference in the positive z direction (in cm).

Usage: change_ref(fname_base, phis, freqs, refs)

Arguments:

fname_base:	the base filename, does not include the _p#, _f#, or .mat
phis:	list of planes (phi values) to phase shift data for; filenames are derived from this
freqs:	list of frequencies to evaluate (GHz) ; filenames are derived from this
refs:	list of reference points to re-evaluate using; each reference value corresponds to a single frequency (in cm)

File Outputs:

creates new files with the same name as the old files with '_r#.mat' appended to represent the new reference

Sample calls for the project named ‘test’:

To change the reference of the frequencies 2.2, 8.4, and 14 GHz using the respective reference points: 7.0, 7.4, 8.1 cm. Make the changes only for the principal planes:

```
change_ref(‘test_ant’, [0 90], [2.2 8.4 14], [7.0 7.4 8.1])
```

This script operates based on the following pseudo-code:

```
for freq in freqs
    for phi in phis
        load data from file for given phi and freq
        change reference of <Etheta
        change reference of <Ephi
        create new output file
```

The pseudo-code for the function which changes the reference is:

```
unwrap phases
removed phase discontinuity in CST MWS data
add a large multiple of 360 to phases to put it in the far field
phase shift using algorithm below
adjust phase to be within [-180, 180)
```

The phase is shifted using a triangle model, where we are solving for the missing side length of the triangle. For this model, all distances are measured as a phase in degrees. Imagine the original reference (origin) is one corner of the triangle. Another point comes from the far-field data; it is at a distance of the phase (φ) and at an angle of the elevation angle (θ). The third vertex of the triangle is the new reference, it is on the z-axis, but offset from the old reference

(origin) by the physical distance (as a phase) between the old reference and the new reference (φ_{pc}). Therefore, the phase shift (φ_{shift}) is the distance between the point off the z-axis and the new reference.

$$\varphi_{shift} = \sqrt{(\varphi \sin\theta - 0)^2 + (\varphi \cos\theta - \varphi_{pc})^2}$$

The following pages contain the source code for change_ref.m.

change_ref.m 9/19/2007

```
function change_ref(varargin)
% takes a data file and phase shifts all of the phase data to represent
% a new reference point (phase center)
% a separate reference will be used for each frequency, and several planes
% (phi values) can be evaluated at each frequency
% the value of the reference is the diplacement from the orginal reference
% in the positive z direction (in cm)
%
% usage: change_ref(fname_base, phis, freqs, refs)
%
% INPUTS:
% fname_base: the base filename, does not include the _p#, _f#, or .mat
% phis: list of planes (phi values) to phase shift data for; filenames are
%       derived from this
% freqs: list of frequencies to evaluate; filenames are derived from this
% refs: list of reference points to re-evaluate using; each reference value
%       corresponds to a single frequency
% OUTPUTS:
% creates new files with the same name as the old files with '_r#.mat'
% appended to represent the new reference
%
% File Name:    change_ref.m
% Author:       Gabe Cohn
%               Caltech, RF and Microwave Group
% Dates:        08-09-2007
% Revisions:    09-10-2007 - Gabe Cohn      - added CST phase shift

assert(nargin == 4, 'ERROR: this function exactly 4 arguments');

fname_base = varargin{1};
phis = varargin{2};
freqs = varargin{3};
refs = varargin{4};

% check args
assert(ischar(fname_base), 'ERROR: the first argument (fname_base) must be a string');
assert(isvector(phis), 'ERROR: the second argument (phis) must be a vector');
assert(~isempty(phis), 'ERROR: the second argumnet (phis) is empty');
assert(isvector(freqs), 'ERROR: the third argument (freqs) must be a vector');
assert(~isempty(freqs), 'ERROR: the third argumnet (freqs) is empty');
assert(isvector(refs), 'ERROR: the fourth argument (refs) must be a vector');
assert(length(freqs) == length(refs), 'ERROR: freqs and refs must be the same length');
files = dir([fname_base,'*']);
assert(~isempty(files), ['ERROR: no files found that match ', fname_base]);

disp('Created the follcwing files:');

for index = 1:length(freqs)
    f_string = num2str(freqs(index));

    ref_phase = 360*refs.*freqs*1e-2*1e9/2.9979e8; % get refs in phase (deg)
    for phi = phis
        filename = [fname_base,'_p',sprintf('%03d',phi),'_f',f_string,'.mat'];
        found = 0;
```

-1-

change_ref.m 9/19/2007

```
55          for i = 1:length(files)
56              if strmatch(files(i).name, filename, 'exact')
57                  found = 1;
58              end
59          end
60          if found == 0
61              disp(['file ', filename, ' not found']);
62              continue
63          end
64          data = load('-ascii', filename); % load data set
65
66          data(:,3) = rerefPhase(data(:,3), data(:,1), ref_phase(index)); % <Etheta
67          data(:,5) = rerefPhase(data(:,5), data(:,1), ref_phase(index)); % <Ephi
68
69          ref_string = num2str(refs(index));
70          ref_string = strrep(ref_string, '.', '_'); % replace '.' with '_'
71          outfile = fopen([fname_base,'_p',sprintf('%03d',phi),'_f',f_string,'_r',
    ref_string,'.mat'], 'w');
72          for i = 1:size(data,1)
73              fprintf(outfile, '%f\t%e\t%f\t%e\t%f\n', data(i,1), data(i,2), data(i,3),
    data(i,4), data(i,5));
74          end
75          fclose(outfile);
76          disp(['    ',fname_base,'_p',sprintf('%03d',phi),'_f',f_string,'_r',ref_string,
    '.mat']);
77          clear data; % clear data from memory
78      end
79  end
80  return
81
82  function newPhases = rerefPhase(oldPhases, thetas, ref_phase)
83  % all values in degrees , including the ref which needs to be in  a phase
84  assert(length(oldPhases) == length(thetas), 'phases and thetas must have the same
    length');
85
86  oldPhases = 180/pi*unwrap(pi/180*oldPhases); % unwrap the phases (in degrees)
87
88  % CST always puts a phase discontinuity between theta=-1 and theta=0
89  % I don't know why this is, but this code should remove that discontinuity
90  mid = find(thetas == 0, 1);
91  if abs(round(oldPhases(mid-1)-oldPhases(mid))) == 180
92     oldPhases(1:mid-1) = oldPhases(1:mid-1)-round(oldPhases(mid-1)-oldPhases(mid));
93  end
94
95  oldPhases = oldPhases + 1e5*360; % put them all in the far field
96      % 1e5 is just a big number, the greater this number the better the fit,
97      % at the expense of computation time
98
99  newPhases = zeros(size(oldPhases));
100 for i = 1:length(thetas)
101     newPhases(i) = getEquiPhase(ref_phase, thetas(i), oldPhases(i));
102 end
103 % put the center point within [-180,180) with same relative phase
104 newPhases = newPhases + 180;
```

-2-

change_ref.m 9/19/2007

```
105   newPhases = newPhases - 360*floor(newPhases(flocr(length(newPhases)/2)+1)/360);
106   newPhases = newPhases - 180;
107   assert(newPhases(floor(length(newPhases)/2)+1) >= -180 && newPhases(floor(length(
      newPhases)/2)+1) <= 180, 'phase shift error, need to debug');
108   return;
109
110   %%% copied from phase_center.m, Gabe Cohn, 07-19-2007
111   function equiPhase = getEquiPhase(z_pc_phase, theta, phase)
112   % given (theta, phase) from the origin, it returns the phase to the same
113   % point from a new reference (guessed phased center) that is on the
114   % vertical axis, but displaced from the origin by z_pc_phase (in degrees)
115   % all angles are in degrees
116   edge_pt = phase * [sind(theta), cosd(theta)];
117   equiPhase = norm(edge_pt - [0, z_pc_phase]); % distance (in phase) between
118                                 % the edge_pt and the guessed phase center
119   return
120
```

5.3.3 auto_pc.m

The Matlab script auto_pc.m finds the phase center of an antenna based on the far-field patterns for several frequencies using a best fit over all planes listed. In addition, can run new data files with the reference shifted to the phase center (using change_ref.m).

Usage: auto_pc(fname_base, phis, freqs, [phaseShift_phis])

Arguments:

fname_base: the base filename, does not include the _p#, _f#, or .mat
phis: list of planes (phi values) to evaluate; filenames are derived from this
freqs: list of frequencies to evaluate (GHz); filenames are derived from this
phaseShift_phis: (optional) list of phis to produce data files that represent the phase shifted data with the phase center as the reference. If omitted, then no extra files will be produced

File Outputs:

creates a new file named <fname_base>_pc.txt which contains phase center data at all frequencies, with uncertainties
if phaseShift_phis is specified, then new data files will be created with a shifted reference

Sample calls for the project named 'test':

To compute the phase center for every frequency between 1 and 18 GHz in 0.5 GHz steps, using a phi resolution of 15 degrees:

```
auto_pc('test_ant', 0:15:179:, 1:0.5:18)
```

To compute the phase center for every frequency between 1 and 18 GHz in 0.5 GHz steps, using a phi resolution of 15 degrees. In addition, create new data files with shifted phase data for only the principal planes:

```
auto_pc('test_ant', 0:15:179:, 1:0.5:18, [0 90])
```

This script operates based on the following pseudo-code:

```
create output file
for freq in freqs
    for phi in phis
        load data from file for given phi and freq
        store phase data from dominant polarization
    compute phase center using phase_center.m
    write data to output file
    if phaseShift_phis is defined
        create new data files with shifted data using change_ref.m
```

The following pages contain the source code for auto_pc.m.

auto_pc.m 9/19/2007

```
function auto_pc(varargin)
% finds the phase center of an antenna based on the far-field patterns for
% several frequencies using a best fit over all planes listed
%
% usage: auto_pc(fname_base, phis, freqs, [phaseShift_phis])
%
% INPUTS:
% fname_base: the base filename, does not include the _p#, _f#, or .mat
% phis: list of planes (phi values) to evaluate; filenames are derived from
%       this
% freqs: list of frequencies to evaluate; filenames are derived from this
% phaseShift_phis: (optional) list of phis to produce data files that
%   represent the phase shifted data with the phase center as the reference
% OUTPUTS:
% creates a new file named fname_base+'_pc.txt' which contains phase
% center data at all frequencies, with uncertainties
%
% File Name:    auto_pc.m
% Author:       Gabe Cohn
%               Caltech, RF and Microwave Group
% Dates:        07-19-2007; 08-08-2007

assert(nargin == 3 || nargin == 4, 'ERROR: this function takes 3 or 4 arguments');

% default values
phaseShift_phis = zeros(1,0);

fname_base = varargin{1};
phis = varargin{2};
freqs = varargin{3};
if nargin == 4
    phaseShift_phis = varargin{4};
end

% check args
assert(ischar(fname_base), 'ERROR: the first argument (fname_base) must be a string');
assert(isvector(phis), 'ERROR: the second argument (phis) must be a vector');
assert(isvector(freqs), 'ERROR: the third argument (freqs) must be a vector');
assert(~isempty(freqs), 'ERROR: the third argumnet (freqs) is empty');
assert(isvector(phaseShift_phis), 'ERROR: the fourth argument (phaseShift_phis) must be a vector');
files = dir([fname_base,'*']);
assert(~isempty(files), ['ERROR: no files found that match ', fname_base]);

% open output file and write header
outfile = fopen([fname_base,'_pc.txt'], 'w');
fprintf(outfile, '%s Phase Center Data\n\n', fname_base);
fprintf(outfile, 'freq - frequency in GHz\n');
fprintf(outfile, 'z_pc - position of the phase center on the z-axis (in cm)\n');
fprintf(outfile, 'dev - standard deviation of the phase measured from the phase center (in degrees)\n\tThis is a measure of goodness of fit for the phase center\n\n');
for i = 1:length(phis)
    if i == 1
        fprintf(outfile, 'Using the following planes (phi values):\n');
```

auto_pc.m 9/19/2007

```
53          fprintf(outfile, '\t%d', phis(1));
54      else
55          fprintf(outfile, ', %d', phis(i));
56      end
57  end
58  fprintf(outfile, '\n\n freq       z_pc      dev\n\n');
59
60  for freq = freqs
61      f_string = num2str(freq);
62  %     f_string = strrep(f_string, '.', '_'); % replace '.' with '_'
63
64      thetas = 0;
65      phases = zeros(0,0);
66      for phi = phis
67          filename = [fname_base,'_p',sprintf('%03d',phi),'_f',f_string,'.mat'];
68          found = 0;
69          for i = 1:length(files)
70              if strmatch(files(i).name, filename, 'exact')
71                  found = 1;
72              end
73          end
74          if found == 0
75              disp(['file ', filename, ' not found']);
76              continue
77          end
78          data = load('-ascii', filename); % load data set
79
80          if thetas == 0 % then it's the first data set to be loaded
81              thetas = data(:,1);
82          else
83              assert(min(thetas == data(:,1)), 'ERROR: data sets are not compatable');
84          end
85
86          col = getDom(data, [-45.1,44.9]) + 1; % determine which polarization dominates
    (E_theta or E_phi)
87          phases = cat(1, phases, data(:,col)'); % append the phase data of the dominant
    polarization
88
89          clear data; % clear data from memory
90      end
91
92      % get z_pc and dev
93      [z_pc, dev] = phase_center(thetas,phases,freq);
94
95      % store results
96      z_pc = 100*z_pc; % put z_pc in cm
97      disp(sprintf('%4.1f GHz:\tz_pc: %9f cm\tdev: %9f deg', freq, z_pc, dev));
98      fprintf(outfile, '%-10f %-9f %-9f\n', freq, z_pc, dev);
99
100     if ~isempty(phaseShift_phis) % run chnage_ref to create data files for the new
    phase_center
101         change_ref(fname_base, phaseShift_phis, freq, z_pc);
102     end
103 end
```

-2-

auto_pc.m 9/19/2007

```
104
105   fclose(outfile);
106   return
107
108   function col = getDom(data, angle_range)
109   % determines if E_theta or E_phi dominates in the angle range
110   % returns the column number of the one that dominates (2 or 4)
111   % 2 = E_theta; 4 = E_phi
112   assert(size(data, 2) == 5, 'data sets must have 5 columns');
113   assert(isvector(angle_range), 'range must be a vector');
114   assert(length(angle_range) == 2, 'the length of range must be 2');
115
116   angles = data(:,1);
117
118   %%% copied from phase_center.m, Gabe Cohn, 07-19-2007
119   % find indicies corresponding the angle_range
120   angle_index = [0,length(angles)]; % default to using entire range
121   i = 1;
122   while (i <= length(angles)) && (angles(i) < angle_range(1))
123       i = i + 1;
124   end
125   if i > length(angles)
126       disp('ERROR: all angles are less than the min range value');
127       disp('will use the entire data set');
128   else
129       angle_index(1) = i;
130       while (i <= length(angles)) && (angles(i) <= angle_range(2))
131           i = i + 1;
132       end
133       if i < angle_index(2)
134           angle_index(2) = i;
135       end
136   end
137
138   Etheta_sum = 0;
139   Ephi_sum = 0;
140   for i = angle_index(1):angle_index(2)
141       Etheta_sum = Etheta_sum + data(i, 2);
142       Ephi_sum = Ephi_sum + data(i, 4);
143   end
144
145   if Etheta_sum > Ephi_sum
146       col = 2;
147   else
148       col = 4;
149   end
150   return
151
```

5.4 Computation of Radiated Power, Taper and Spillover Efficiencies

5.4.1 eff.m

The Matlab script eff.m computes spillover efficiency, taper efficiency, and power radiated of the feed based on the far-field pattern. The current approximation assumes that the E-field changes linearly between the planes of input data (phi slices).

Usage: eff(fname_base, phis, freqs)

Arguments:

fname_base: the base filename, does not include the _p#, _f#, or .mat
phis: list of planes (phi values) to evaluate; filenames are derived from this
freqs: list of frequencies to evaluate (GHz); filenames are derived from this

File Outputs:

<fname_base>_taper.txt - contains taper (illumination) efficiency versus theta (subtended half-angle of elevation) for each frequency
<fname_base>_spill.txt - contains spillover efficiency versus theta (subtended half angle of elevation) for each frequency
<fname_base>_eff.txt - contains the product of the spillover efficiency and taper efficiency versus theta (subtended half angle of elevation) for each frequency
<fname_base>_pow.txt - contains radiated power data versus theta (subtended half angle of elevation) for each frequency

Sample calls for the project named ‘test’:

To compute the efficiencies for every frequency between 1 and 18 GHz in 0.5 GHz steps, using a phi resolution of 15 degrees for the slices:

```
auto_pc(‘test_ant’, 0:15:179:, 1:0.5:18)
```

This script operates based on the following pseudo-code:

```
for freq in freqs
    for phi in phis
        load data from file for given phi and freq
        save gain {G(theta, phi)}
    reshape gain so that theta in [0,180] and phi in [0, 360)
    compute gain as a function of only theta
    compute power radiated at all angles greater than theta as a function of theta
    compute taper efficiency
    compute spillover efficiency
    compute taper and spillover efficiency product
    store results for the given freq
write output files
```

This script uses the calculations found on the next page.

First, the power gain is calculated as a function of theta and phi from the far-field patterns:

$$G(\theta, \phi) \propto |E|^2 = |E_\theta|^2 + |E_\phi|^2$$

Then, the gain is expressed as a function of only theta, by taking the average over all phi slices:

$$G(\theta) = \frac{1}{2\pi} \int_0^{2\pi} G(\theta, \phi)\, d\phi$$

The data is not continuous, so the transition between the elevation angles is assumed to be linear:

$$\int_0^{2\pi} G(\theta, \phi)\, d\phi = \sum_i \int_{\phi_i}^{\phi_{i+1}} \left(\frac{G(\theta, \phi_{i+1}) - G(\theta, \phi_i)}{\phi_{i+1} - \phi_i} (\phi - \phi_i) + G(\theta, \phi_i) \right) d\phi$$

From the gain function, the power radiated as a function of subtended half-angle is calculated. This function represents all power radiated within the elevation (theta) angles of $[-\theta_0, \theta_0]$:

$$p(\theta_0) \equiv \int_0^{\theta_0} G(\theta) \sin\theta\, d\theta$$

This function is used in the calculation of both the taper efficiency and the spillover efficiency. However, once again the data is not continuous, and therefore the transition between the elevation angles is assumed to be linear:

$$p(\theta_0) \equiv \int_0^{\theta_0} G(\theta) \sin\theta\, d\theta = \sum_i \int_{\theta_i}^{\theta_{i+1}} \left(\frac{G(\theta_{i+1}) - G(\theta_i)}{\theta_{i+1} - \theta_i} (\theta - \theta_i) + G(\theta_i) \right) \cdot \sin\theta\, d\theta$$

On page 813, Balanis [5-2] defines the taper efficiency to be the "uniformity of the amplitude distribution of the feed pattern over the surface of the reflector." Balanis [5-2] gives the equation for the taper efficiency on page 815:

$$\epsilon_t(\theta_0) = 2 \cot^2\left(\frac{\theta_0}{2}\right) \frac{\left| \int_0^{\theta_0} \sqrt{G(\theta)}\, \tan\left(\frac{\theta}{2}\right) d\theta \right|^2}{\int_0^{\theta_0} G(\theta)\, \sin\theta\, d\theta}$$

This can be simplified by expressing it in terms of the radiated power function ($p(\theta_0)$):

$$\epsilon_t(\theta_0) = \frac{2}{p(\theta_0)} \cot^2\left(\frac{\theta_0}{2}\right) \left| \int_0^{\theta_0} \sqrt{G(\theta)}\, \tan\left(\frac{\theta}{2}\right) d\theta \right|^2$$

On page 813, Balanis [5-2] defines the spillover efficiency to be the "fraction of the power that is radiated by the feed, intercepted, and collimated by the reflector." Balanis [5-2] gives the equation for the spillover efficiency on page 815. This too can be simplified by expressing it in terms of the radiated power function ($p(\theta_0)$):

$$\epsilon_s(\theta_0) = \frac{\int_0^{\theta_0} G(\theta) \sin\theta\, d\theta}{\int_0^{\pi} G(\theta) \sin\theta\, d\theta} = \frac{p(\theta_0)}{p(\pi)}$$

Since these two efficiencies complete with each other, it can be useful to plot their product:

$$\epsilon_{prod}(\theta_0) = \epsilon_t(\theta_0)\, \epsilon_s(\theta_0)$$

The following pages contain the source code for eff.m.

eff.m 9/19/2007

```
function eff(varargin)
% computes spillover efficiency, taper efficiency, and power radiated based
% on the far field pattern of an antenna.
% current approximation assumes that the E-field changes linearly between
% the planes of input data (phi values)
%
% usage: eff(fname_base, phis, freqs)
%
% INPUTS:
% fname_base: the base filename, does not include the _p#, _f#, or .mat
% phis: list of planes (phi values) to evaluate; filenames are derived from
%       this
% freqs: list of frequencies to evaluate; filenames are derived from this
% OUTPUTS:
% creates the following new files:
%   fname_base+'_taper.txt' - contains taper (illumination) efficiency
%       versus theta (subtended half-angle of elevation) for each frequency
%   fname_base+'_spill.txt' - contains spillover efficiency versus theta
%       (subtended half angle of elevation) for each frequency
%   fname_base+'_eff.txt' - contains the product of the spillover
%       efficiency and taper efficiency for each frequency
%   fname_base+'_pow.txt' - contains radiated power data versus theta
%       (subtended half angle of elevation) for each frequency
%
% File Name:    eff.m
% Author:       Gabe Cohn
%               Caltech, RF and Microwave Group
% Dates:        08-13-2007; 08-15-2007 -- 08-17-2007

assert(nargin == 3, 'ERROR: this function exactly 3 arguments');

fname_base = varargin{1};
phis = varargin{2};
freqs = varargin{3};

% check args
assert(ischar(fname_base), 'ERROR: the first argument (fname_base) must be a string');
assert(isvector(phis), 'ERROR: the second argument (phis) must be a vector');
assert(length(phis) >= 2, 'ERROR: the second argumnet (phis) must specify at least 2 values');
assert(isvector(freqs), 'ERROR: the third argument (freqs) must be a vector');
assert(~isempty(freqs), 'ERROR: the third argumnet (freqs) is empty');

files = dir([fname_base,'*']);
assert(~isempty(files), ['ERROR: no files found that match ', fname_base]);

thetas = 0;
phis_old = phis; % make a copy
taper_outdata = zeros([0 0]); % matrix for taper eff. output data
spill_outdata = zeros([0 0]); % matrix for spillover output data
prod_outdata = zeros([0 0]); % matrix for product of spill and taper output data
pow_outdata = zeros([0 0]); % matrix for power output data
for freq = freqs
    f_string = num2str(freq);
```

-1-

eff.m 9/19/2007

```
54
55      thetas = 0;
56      phis = phis_old;
57      G_theta_phi = zeros(0,0);
58      for phi = phis
59          filename = [fname_base,'_p',sprintf('%03d',phi),'_f',f_string,'.mat'];
60          found = 0;
61          for i = 1:length(files)
62              if strmatch(files(i).name, filename, 'exact')
63                  found = 1;
64              end
65          end
66          if found == 0
67              disp(['file ', filename, ' not found']);
68              continue
69          end
70          data = load('-ascii', filename); % load data set
71
72          if thetas == 0 % then it's the first data set to be loaded
73              thetas = data(:,1);
74          else
75              assert(min(thetas == data(:,1)), 'ERROR: data sets are not compatable');
76          end
77
78          G_tmp = zeros(size(thetas));
79          for i = 1:size(G_tmp)
80              G_tmp(i,1) = norm([data(i,2), data(i,4)]).^2; % get |E|^2 using |Etheta| ↵
    and |Ephi|
81          end
82          G_theta_phi = cat(2, G_theta_phi, G_tmp); % append the power gain data
83
84          clear data; % clear data from memory
85      end
86
87      % reshape G_theta_phi matrix so: theta in [0,180]; phi in [0,360)
88      G_old = G_theta_phi; % make a copy
89      phis_old = phis; % make a copy
90      G_theta_phi = zeros([floor(size(G_old,1)/2)+1, 2*size(G_theta_phi,2)]);
91      zero_theta = find(thetas == 0);
92      thetas = thetas(zero_theta:length(thetas));
93      phis = cat(2, phis, 180+phis);
94      G_theta_phi(:,1:size(G_old,2)) = G_old(zero_theta:size(G_old,1),:);
95      for i = 1:zero_theta
96         G_theta_phi(i,(size(G_old,2)+1):size(G_theta_phi,2)) = G_old(zero_theta+1-i,:);
97      end
98
99      G_theta = integPhi(phis, G_theta_phi)/360; % average over all phi planes
100
101     P_theta = zeros([length(thetas), 1]);
102     for i = 2:length(thetas) % skip i = 1 (theta=0); P_theta(i=1) = 0
103         P_theta(i) = integTheta(thetas(1:i),G_theta(1:i));
104     end
105
106     taper_eff = 100*ones([length(thetas) 1]);
```

-2-

eff.m 9/19/2007

```
107         for i = 2:length(thetas) % skip i = 1 (theta=0); taper_eff(i=1) = 100%
108             taper_eff(i) = 100*2*cotd(thetas(i)/2)^2*taperEffNum(thetas(1:i),sqrt(G_theta(1
    :i)))/P_theta(i); % taper eff. in percent
109         end
110         assert(min(taper_eff <= 100), 'taper_eff out of range!');
111
112         spill_eff = 100 * P_theta ./ P_theta(length(P_theta)); % spillover eff. in percent
113
114         taper_outdata = cat(2, taper_outdata, taper_eff);
115         spill_outdata = cat(2, spill_outdata, spill_eff);
116         prod_outdata = cat(2, prod_outdata, spill_eff.*taper_eff./100);
117         pow_outdata = cat(2, pow_outdata, P_theta(length(P_theta)) - P_theta);
118     end
119
120     % create output files
121     writeTaperOutfile(fname_base, freqs, thetas, phis, taper_outdata);
122     writeSpillOutfile(fname_base, freqs, thetas, phis, spill_outdata);
123     writeProdOutfile(fname_base, freqs, thetas, phis, prod_outdata);
124     writePowOutfile(fname_base, freqs, thetas, phis, pow_outdata);
125     return
126     end
127
128     % integration functions
129
130     function y = linTrans(x, x0,x1,y0,y1)
131     % creates a linear transition from y0 to y1 over x0 to x1
132     y = (y1-y0).*(x-x0)./(x1-x0) + y0;
133     end
134
135     function G_theta = integPhi(phis, G)
136     % integrates each row to produce a matrix that it reduced to one column
137     % assuming a linear blend between the known data points
138     % phis are in degrees (although is doesn't matter)
139     % overall function [LaTeX]: \[ \int_0^{2\pi} G(\theta, \phi) \: d\phi \]
140     % actual integral computed [LaTeX]:
141     %   \[\sum_i \int_{\phi_i}^{\phi_{i+1}} \left[\frac{G(\theta, \phi_{i+1}) -
142     %   G(\theta, \phi_{i})}{\phi_{i+1} - \phi_i} (\phi - \phi_i) +
143     %   G(\theta,\phi_i) \right] d\phi \]
144     assert(size(G,2) == length(phis), 'phis and G are inconsistant in dimension');
145         function y = integrand(x)
146             % x in radians (although it doesn't matter, b/c there's no trig)
147             y = linTrans(x, x0,x1,y0,y1);
148         end
149     G_theta = zeros([size(G,1), 1]);
150     for row = 1:length(G_theta)
151         sum = 0;
152         for i = 1:length(phis)-1
153             % set constants for integration
154             x0 = phis(i)*pi/180; % put in radians
155             x1 = phis(i+1)*pi/180; % put in radians
156             y0 = G(row,i);
157             y1 = G(row,i+1);
158             sum = sum + quad(@integrand, x0, x1); % do numerical integration
159         end
```

-3-

eff.m 9/19/2007

```
160         G_theta(row,1) = sum;
161     end
162     end
163
164     function P_theta = integTheta(thetas, G)
165     % integrates the whole vector to get the power radiated over the angles
166     % listed in thetas.  Essentailly calculated the total radiated power
167     % assuming a linear blend between the known data points
168     % thetas must be in degrees!
169     % overall function [LaTeX]: \[ \int_{\theta_0}^{\theta_1} G(\theta) \:
170     %   \sin \theta \: d\theta \]
171     % actual integral computed [LaTeX]:
172     %   \[ \sum_i \int_{\theta_i}^{\theta_{i+1}} \left[\frac{G(\theta_{i+1}) -
173     %   G(\theta_{i})}{\theta_{i+1} - \theta_i} (\theta - \theta_i) +
174     %   G(\theta_i) \right] \sin \theta \: d\theta \]
175     assert(isvector(G), 'G must be a vector');
176     assert(isvector(thetas), 'thetas must be a vector');
177     assert(length(G) == length(thetas), 'thetas and G are inconsistant in dimension');
178         function y = integrand(x)
179             % x in radians
180             y = linTrans(x, x0,x1,y0,y1).*sin(x);
181         end
182     P_theta = 0;
183     for i = 1:length(thetas)-1
184         % set constants for integration
185         x0 = thetas(i)*pi/180; % put in radians
186         x1 = thetas(i+1)*pi/180; % put in radians
187         y0 = G(i);
188         y1 = G(i+1);
189         P_theta = P_theta + quad(@integrand, x0, x1); % do numerical integration
190     end
191     end
192
193     function num = taperEffNum(thetas, E)
194     % calculates the numerator of the taper efficiency over the range of angles
195     % listed in thetas; this includes the integral
196     % thetas must be in degrees!
197     % overall function [LaTeX]:
198     %   \[ \left| \int_0^{\theta_0} \!\! E(\theta) \: \tan \!
199     %   \left(\frac{\theta}{2}\right) d\theta \right|^2 \]
200     assert(isvector(E), 'E must be a vector');
201     assert(isvector(thetas), 'thetas must be a vector');
202     assert(length(E) == length(thetas), 'thetas and E are inconsistant in dimension');
203         function y = integrand(x)
204             % x in radians
205             y = linTrans(x, x0,x1,y0,y1).*tan(x./2);
206         end
207     integ = 0; % total value of the integral
208     for i = 1:length(thetas)-1
209         % set constants for integration
210         x0 = thetas(i)*pi/180; % put in radians
211         x1 = thetas(i+1)*pi/180; % put in radians
212         y0 = E(i);
213         y1 = E(i+1);
```

eff.m 9/19/2007

```
214         integ = integ + quad(@integrand, x0, x1);
215     end
216     num = abs(integ)^2;
217     end
218
219     % file output functions
220
221     function writeTaperOutfile(fname_base, freqs, thetas, phis, taper_outdata)
222     % creates output file for taper efficiency data named
223     % fname_base+'_taper.txt'
224     outfile = fopen([fname_base,'_taper.txt'], 'w');
225     fprintf(outfile, '%s Taper Efficiency Data\n\n', fname_base);
226     fprintf(outfile, 'All values represent the taper efficiency as a percentage.\n\n');
227     fprintf(outfile, 'The first column contains the row values which are subtended
        half-angles of \n\televation (theta) in degrees down from the zenith\n');
228     fprintf(outfile, 'The first row contains the row values which are frequencies (in
        GHz)\n\n');
229     for i = 1:length(phis)
230         if i == 1
231             fprintf(outfile, 'Using the following planes (phi values):\n');
232             fprintf(outfile, '\t%d', phis(1));
233         else
234             fprintf(outfile, ', %d', phis(i));
235         end
236     end
237     fprintf(outfile, '\n\nTheta \\ Freq    ');
238     for i = 1:length(freqs)
239         fprintf(outfile, '%-16f', freqs(i));
240     end
241     fprintf(outfile, '\n');
242     for i = 1:length(thetas)
243         fprintf(outfile, '%-16d', round(thetas(i)));
244         for j = 1:length(freqs)
245            fprintf(outfile, '%-16f', taper_outdata(i,j));
246         end
247         fprintf(outfile, '\n');
248     end
249     fclose(outfile);
250     end
251
252     function writeSpillOutfile(fname_base, freqs, thetas, phis, spill_outdata)
253     % creates output file for spillover efficiency data named
254     % fname_base+'_spill.txt'
255     outfile = fopen([fname_base,'_spill.txt'], 'w');
256     fprintf(outfile, '%s Spillover Efficiency Data\n\n', fname_base);
257     fprintf(outfile, 'All values represent the spillover efficiency as a percentage.\n\n');
258     fprintf(outfile, 'The first column contains the row values which are subtended
        half-angles of \n\televation (theta) in degrees down from the zenith\n');
259     fprintf(outfile, 'The first row contains the row values which are frequencies (in
        GHz)\n\n');
260     for i = 1:length(phis)
261         if i == 1
262             fprintf(outfile, 'Using the following planes (phi values):\n');
263             fprintf(outfile, '\t%d', phis(1));
```

eff.m 9/19/2007

```
264         else
265             fprintf(outfile, ', %d', phis(i));
266         end
267     end
268     fprintf(outfile, '\n\nTheta \\ Freq    ');
269     for i = 1:length(freqs)
270         fprintf(outfile, '%-16f', freqs(i));
271     end
272     fprintf(outfile, '\n');
273     for i = 1:length(thetas)
274         fprintf(outfile, '%-16d', round(thetas(i)));
275         for j = 1:length(freqs)
276            fprintf(outfile, '%-16f', spill_outdata(i,j));
277         end
278         fprintf(outfile, '\n');
279     end
280     fclose(outfile);
281     end
282
283     function writeProdOutfile(fname_base, freqs, thetas, phis, prod_outdata)
284     % creates output file for the product of spillover and taper efficiencies
285     % the is named fname_base+'_eff.txt'
286     outfile = fopen([fname_base,'_eff.txt'], 'w');
287     fprintf(outfile, '%s Product of Spillover and Taper Efficiencies Data\n\n', fname_base);
288     fprintf(outfile, 'All values represent the product of the spillover efficiency and the
        taper \n\tefficiency as a percentage.\n\tThis is crude form of the aperture
        efficiency\n');
289     fprintf(outfile, 'The first column contains the row values which are subtended
        half-angles of \n\televation (theta) in degrees down from the zenith\n');
290     fprintf(outfile, 'The first row contains the row values which are frequencies (in
        GHz)\n\n');
291     for i = 1:length(phis)
292         if i == 1
293             fprintf(outfile, 'Using the following planes (phi values):\n');
294             fprintf(outfile, '\t%d', phis(1));
295         else
296             fprintf(outfile, ', %d', phis(i));
297         end
298     end
299     fprintf(outfile, '\n\nTheta \\ Freq    ');
300     for i = 1:length(freqs)
301         fprintf(outfile, '%-16f', freqs(i));
302     end
303     fprintf(outfile, '\n');
304     for i = 1:length(thetas)
305         fprintf(outfile, '%-16d', round(thetas(i)));
306         for j = 1:length(freqs)
307            fprintf(outfile, '%-16f', prod_outdata(i,j));
308         end
309         fprintf(outfile, '\n');
310     end
311     fclose(outfile);
312     end
313
```

eff.m 9/19/2007

```
function writePowOutfile(fname_base, freqs, thetas, phis, pow_outdata)
% creates output file for radiated power data named
% fname_base+'_pow.txt'
outfile = fopen([fname_base,'_pow.txt'], 'w');
fprintf(outfile, '%s Radiated Power Data\n\n', fname_base);
fprintf(outfile, 'All values are powers in arbitraty linear units. \nAlthough these powers are in arbitrary units, \n\tthey can be compared to each other (including among different frequencies).  \nThe values represent the power radiated at the given angle and all angles greater. \nTherefore the total radiated power is the value for theta=0.\n\n');
fprintf(outfile, 'The first column contains the row values which are subtended half-angles of \n\televation (theta) in degrees down from the zenith\n');
fprintf(outfile, 'The first row contains the row values which are frequencies (in GHz)\n\n');
for i = 1:length(phis)
    if i == 1
        fprintf(outfile, 'Using the following planes (phi values):\n');
        fprintf(outfile, '\t%d', phis(1));
    else
        fprintf(outfile, ', %d', phis(i));
    end
end
fprintf(outfile, '\n\nTheta \\ Freq    ');
for i = 1:length(freqs)
    fprintf(outfile, '%-16f', freqs(i));
end
fprintf(outfile, '\n');
for i = 1:length(thetas)
    fprintf(outfile, '%-16d', round(thetas(i)));
    for j = 1:length(freqs)
       fprintf(outfile, '%-16e', pow_outdata(i,j));
    end
    fprintf(outfile, '\n');
end
fclose(outfile);
end

```

6

Results

6.1	**Comparison between QuickWave and CST**	**165**
6.1.1	**Return Loss**	**165**
6.1.2	**Far-field Patterns for Feed Only**	**167**
6.1.3	**Far-field Patterns for Feed in Radiation Shield**	**175**
6.1.4	**Far-field Patterns for Feed in Absorber Lined Rad. Shield**	**184**
6.2	**Data from QuickWave Model of Inventor File Design (1-18 GHz)**	**193**
6.2.1	**Far-field Patterns for Feed Only**	**194**
6.2.2	**Far-field Patterns for Feed in Radiation Shield**	**196**
6.2.3	**Far-field Patterns for Feed in Absorber Lined Rad. Shield**	**198**
6.2.4	**Phase Center**	**200**
6.2.5	**Efficiencies for Feed Only**	**201**
6.2.6	**Efficiencies for Feed in Radiation Shield**	**202**
6.2.7	**Efficiencies for Feed in Absorber Lined Rad. Shield**	**203**
6.3	**Data from CST Model of 3164-05 Design (1-18 GHz)**	**204**
6.3.1	**Return Loss**	**204**
6.3.2	**Far-field Patterns for Feed Only**	**206**
6.3.3	**Far-field Patterns for Feed in Radiation Shield**	**215**
6.3.4	**Far-field Patterns for Feed in Absorber Lined Rad. Shield**	**225**
6.3.5	**Phase Center**	**235**
6.3.6	**Efficiencies for Feed Only**	**236**
6.3.7	**Efficiencies for Feed in Radiation Shield**	**237**
6.3.8	**Efficiencies for Feed in Absorber Lined Rad. Shield**	**238**
6.4	**Data from CST Model of 3164-06 Design (0.3-6 GHz)**	**239**
6.4.1	**Return Loss**	**239**
6.4.2	**Far-field Patterns for Feed Only**	**241**
6.4.3	**Far-field Patterns for Feed in Radiation Shield**	**242**
6.4.4	**Far-field Patterns for Feed in Absorber Lined Rad. Shield**	**244**
6.4.5	**Phase Center**	**246**
6.4.6	**Efficiencies for Feed Only**	**247**
6.4.7	**Efficiencies for Feed in Radiation Shield**	**248**
6.4.8	**Efficiencies for Feed in Absorber Lined Rad. Shield**	**249**
6.5	**Effect of Fin Thickness**	**250**
6.5.1	**Far-field Patterns for 3.615 mm Fin**	**251**
6.5.2	**Far-field Patterns for 1.372 mm Fin**	**253**
6.5.3	**Far-field Patterns for 1.372 mm Fin with 3.615 mm at Slot**	**255**
6.5.4	**Far-field Patterns for 3.615 mm Fin with 3.615 mm Edge**	**257**
6.5.5	**Comparison of Far-field Patterns**	**259**

6.1 Comparison between QuickWave and CST

The Inventor file design of the ETS-Lindgren 3164-05 (see section 3.1.1) was modeled and simulated using both QuickWave and CST MWS. The following data shows a comparison between these two models as well as a comparison to measured data. It is important to realize that the Inventor file design is different from the actual feed, from which the measured data was taken. The images below show the actual feed and the Inventor files design side by side. It is clear that the fin shape is different. This is expected to result in discrepancies particularly at the low frequencies.

Lindgren 3164-05 Actual Feed

Figure 6.1 a

Lindgren Inventor File Design Model (CST)

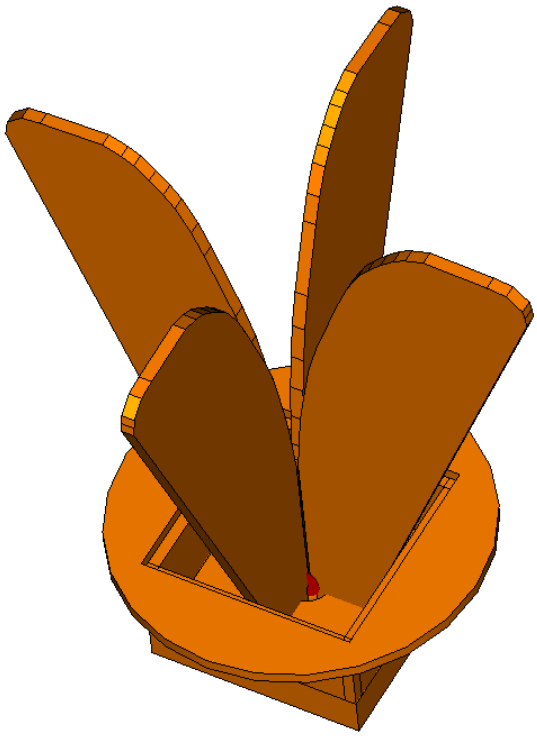
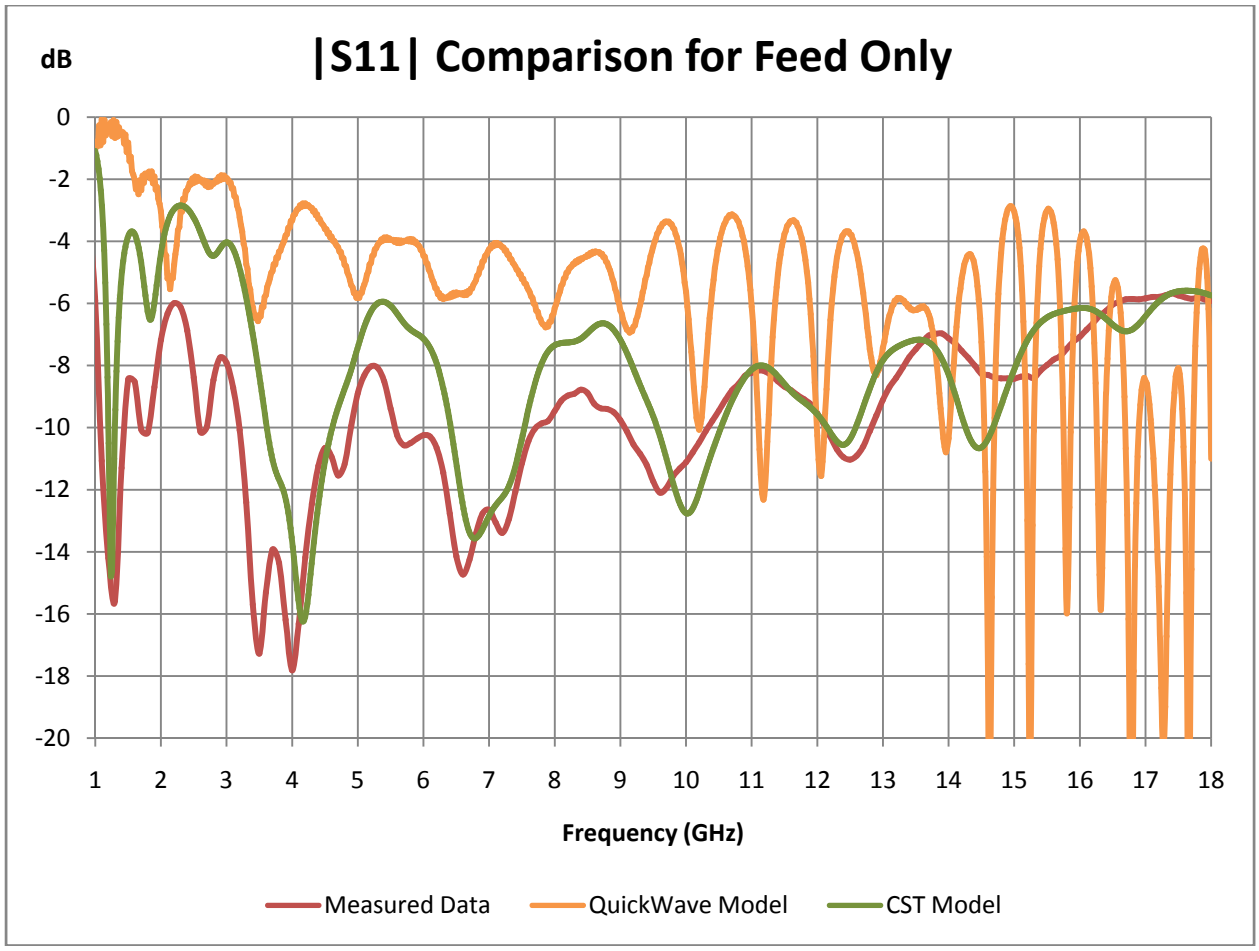


Figure 6.1 b

6.1.1 Return Loss

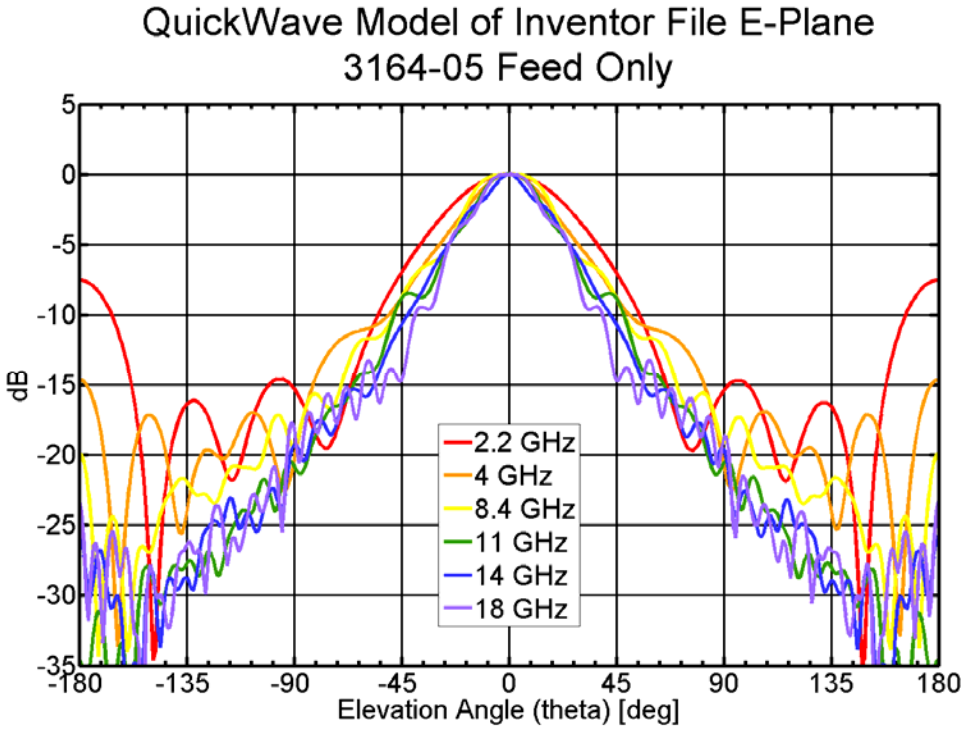
It is important to realize the differences between the models and the actual feed. The input of the actual feed (from which the measured data was taken) is a SMA connector that connects to coax going to a coax-to-slotline transition inside the slot between the fins. Neither the QuickWave nor CST models model this complicated input network. Both models are fed by a simple E-field excitation across the slot at the same point that the coax excitation would be located. This should explain significant differences between the modeled return loss and the measured data, because the models will not reflect losses in the coax or coax-to-slotline transition.

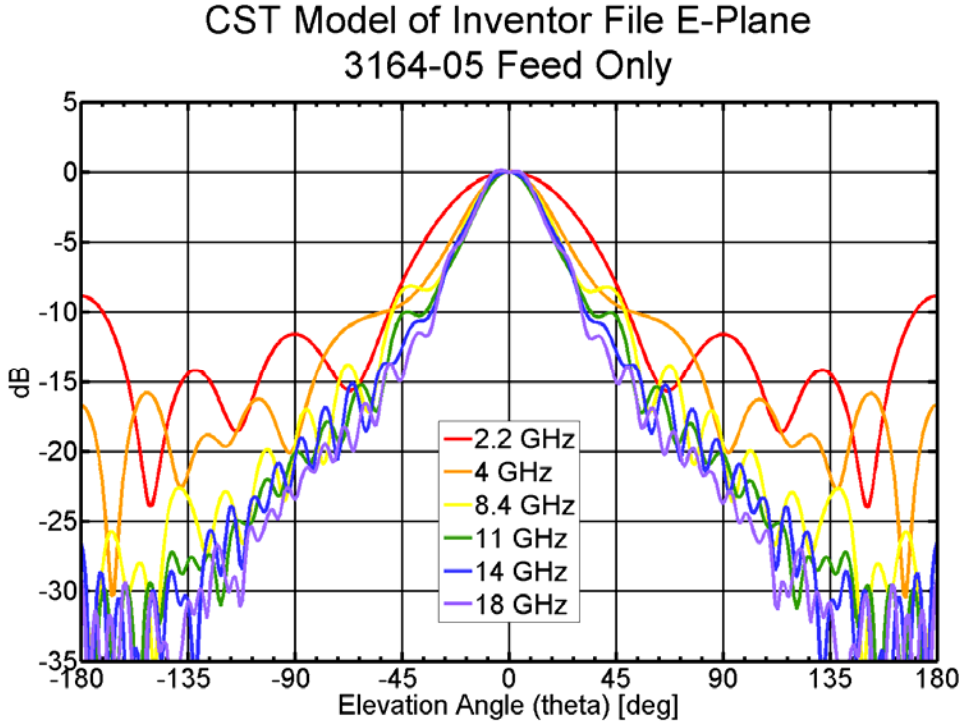


The QuickWave model appears to be a horrible model of the return loss. However, the CST model shows surprising similarity to the measured values.

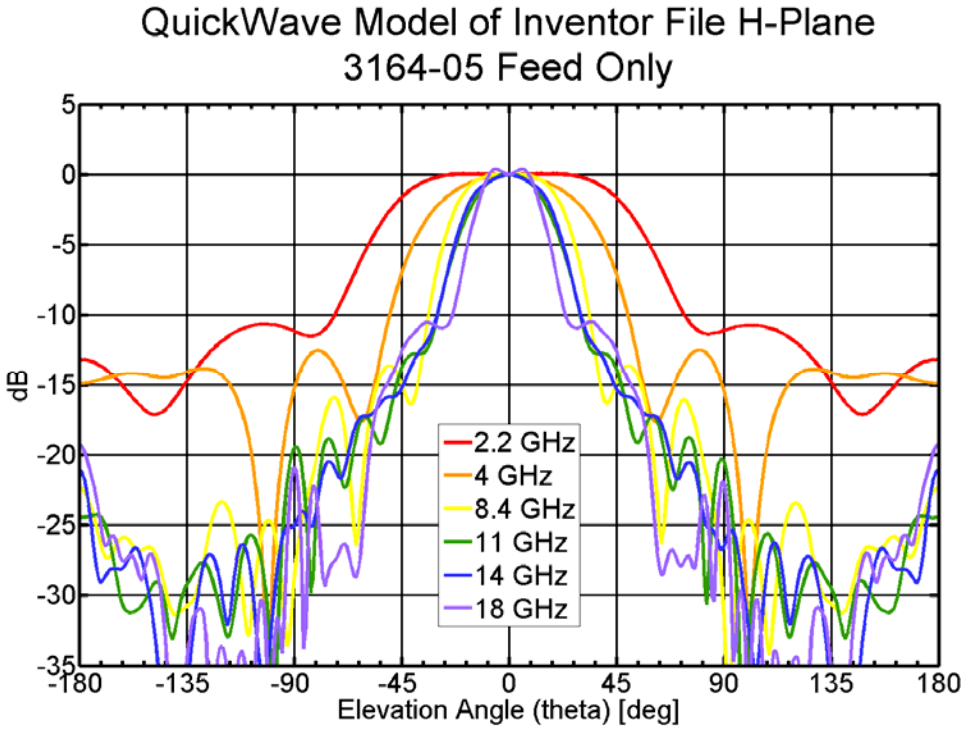
6.1.2 Far-Field Patterns for Feed Only

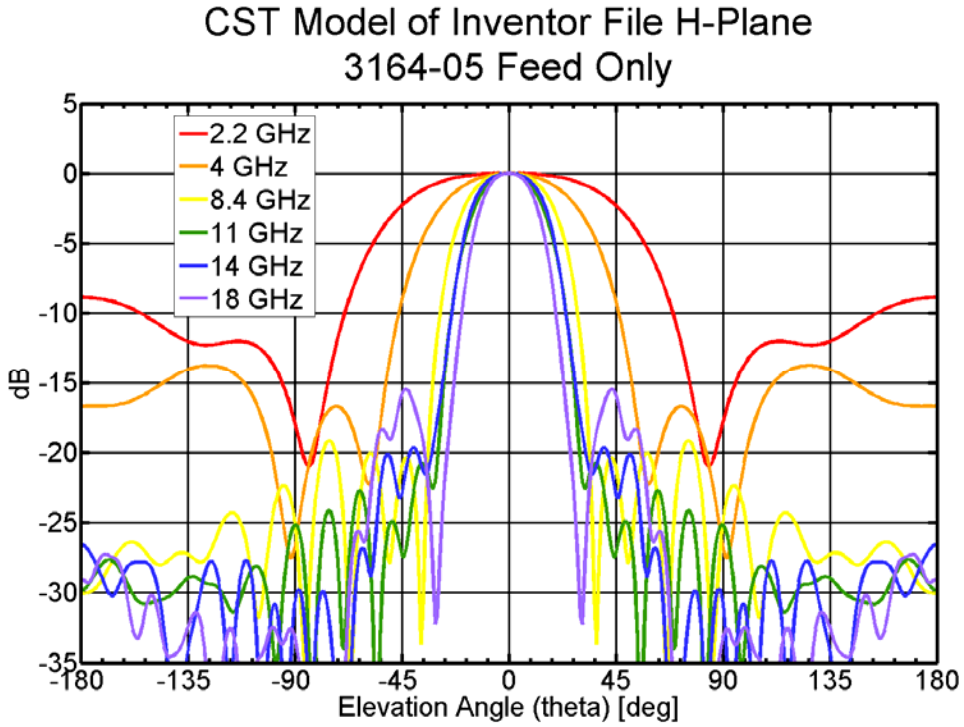
Comparison of QuickWave Model and CST Model E-Plane Patterns:



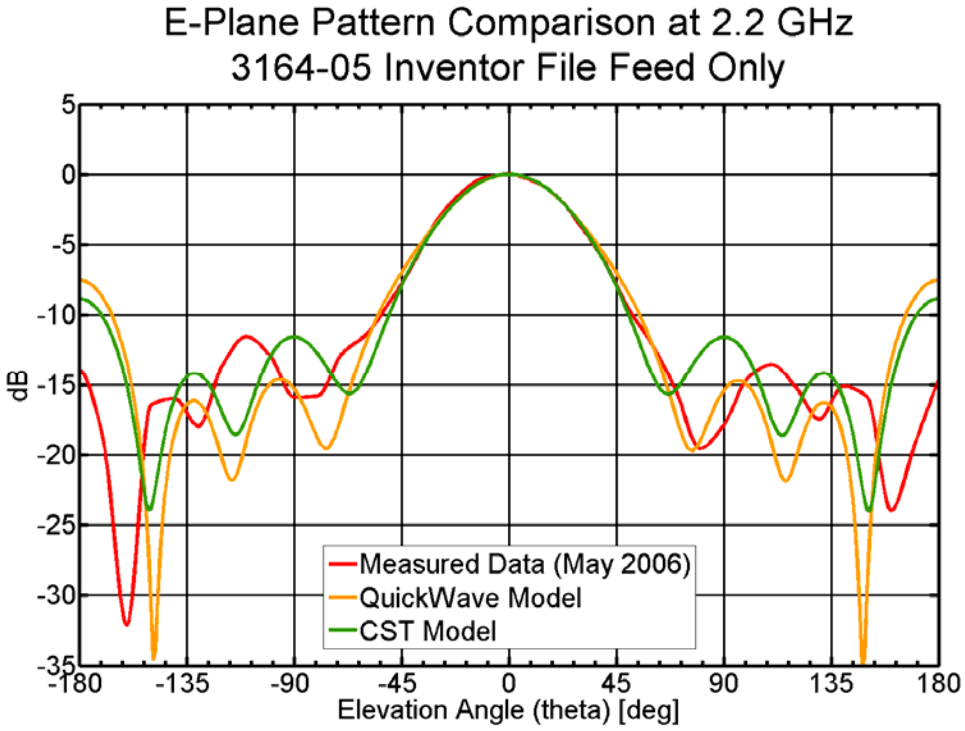


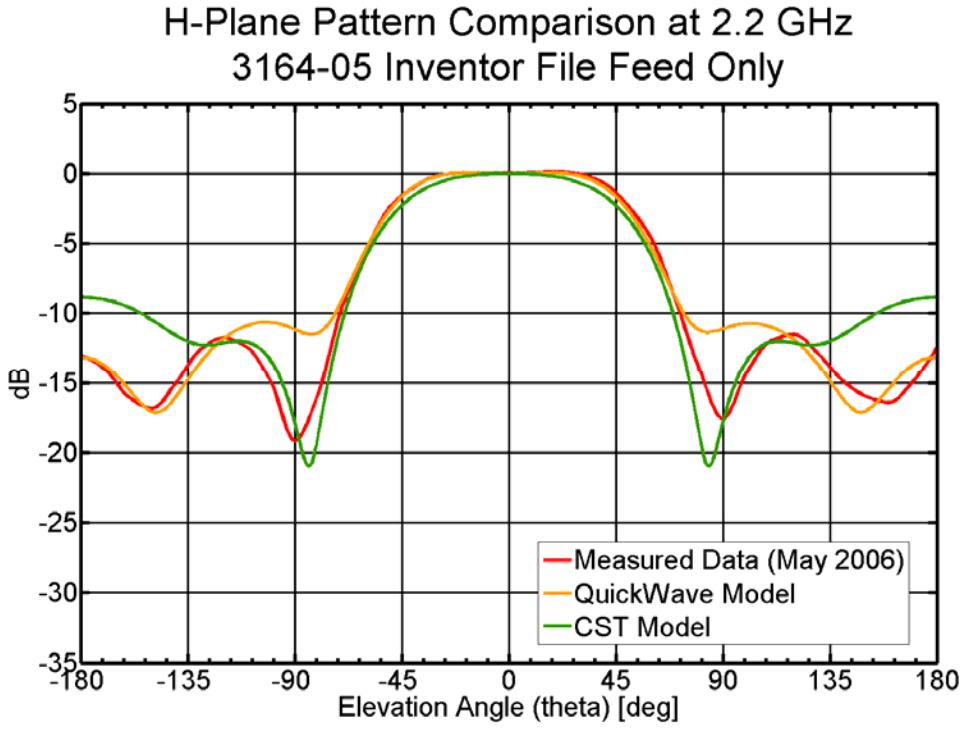
Comparison of QuickWave Model and CST Model H-Plane Patterns:



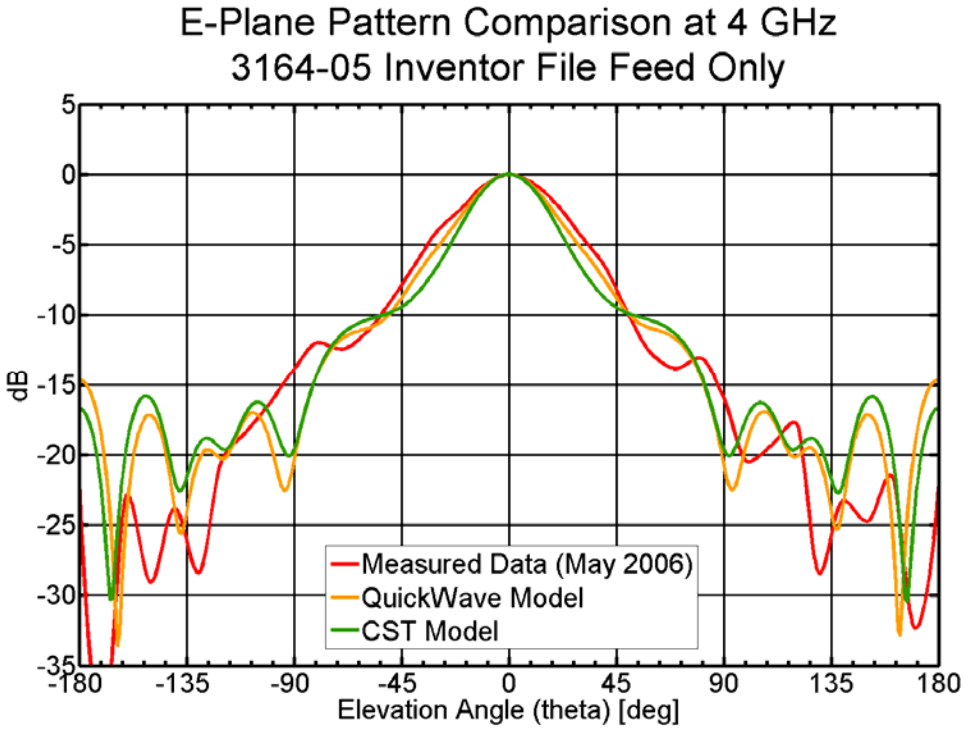


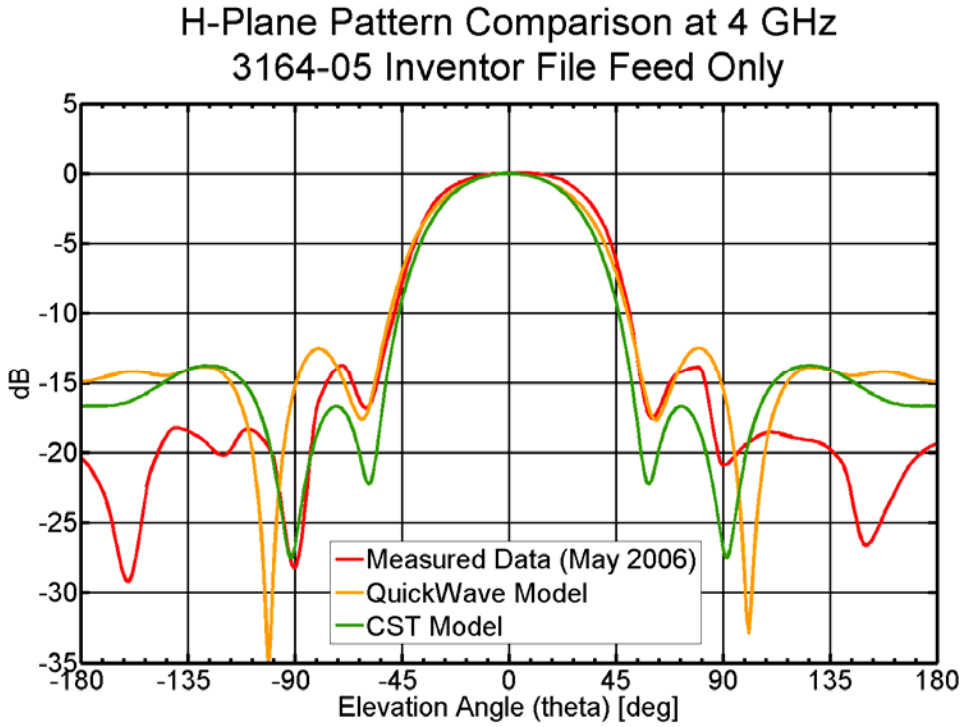
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 2.2 GHz:



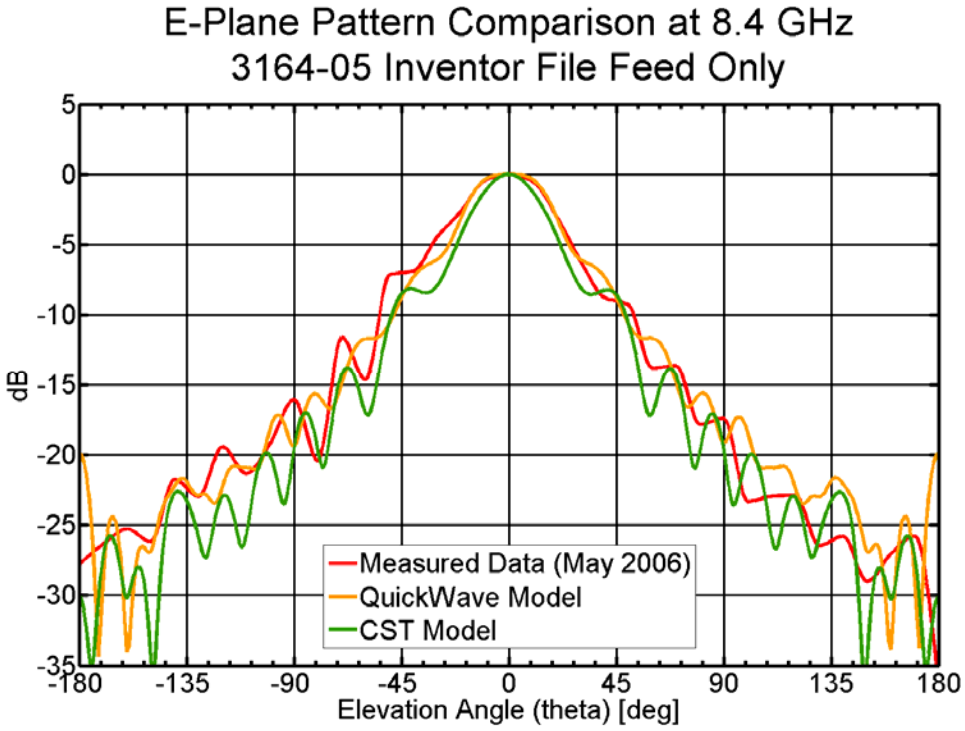


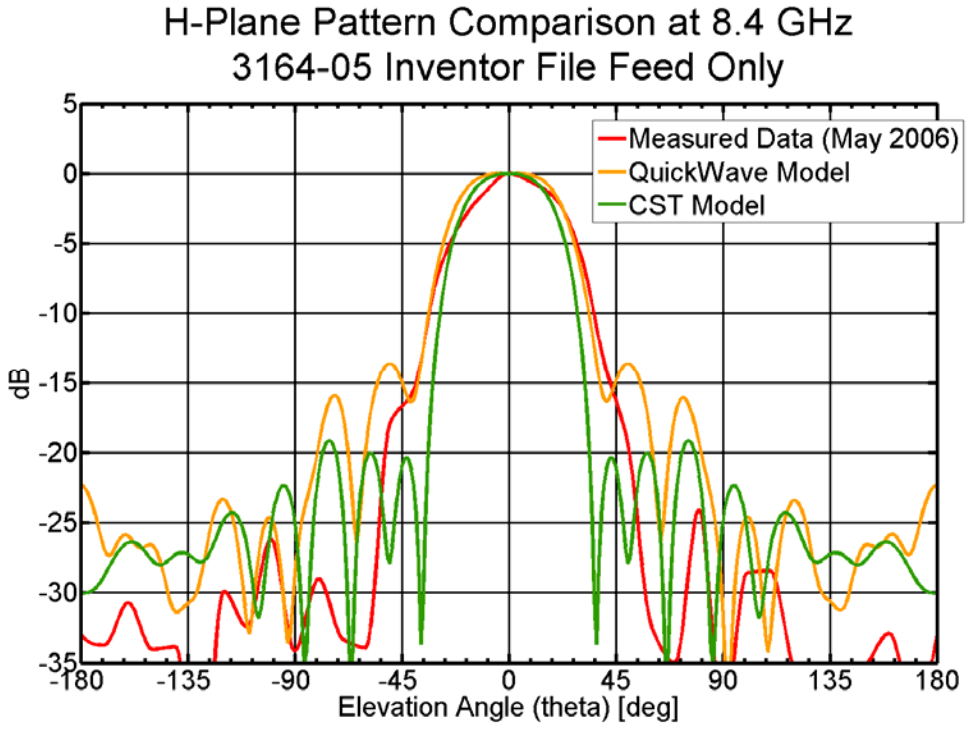
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 4.0 GHz:



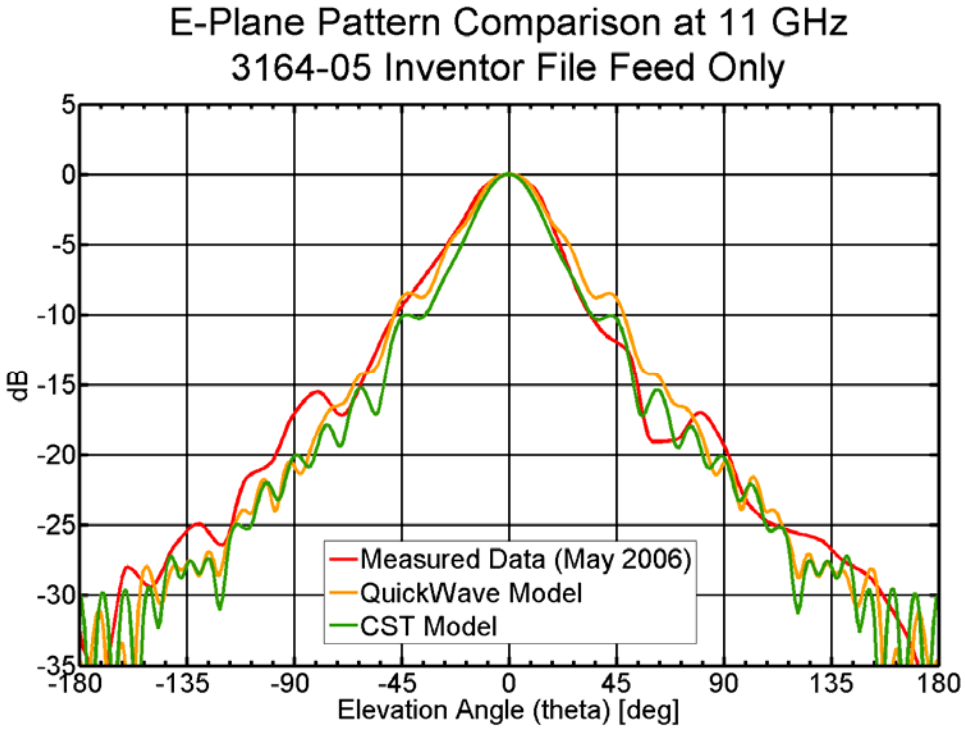


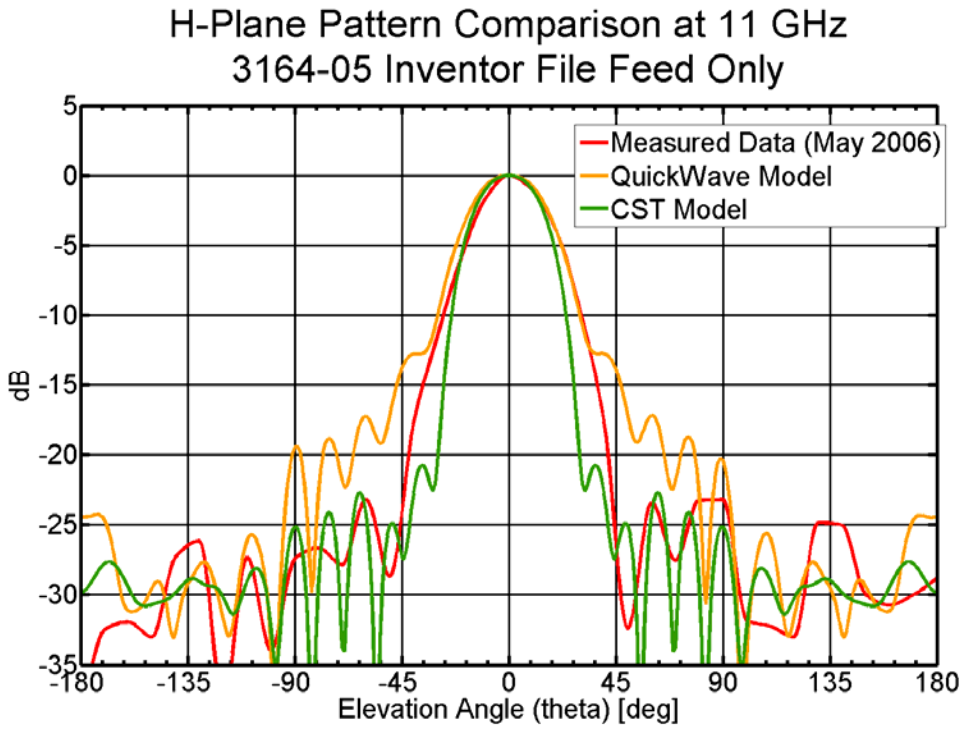
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 8.4 GHz:



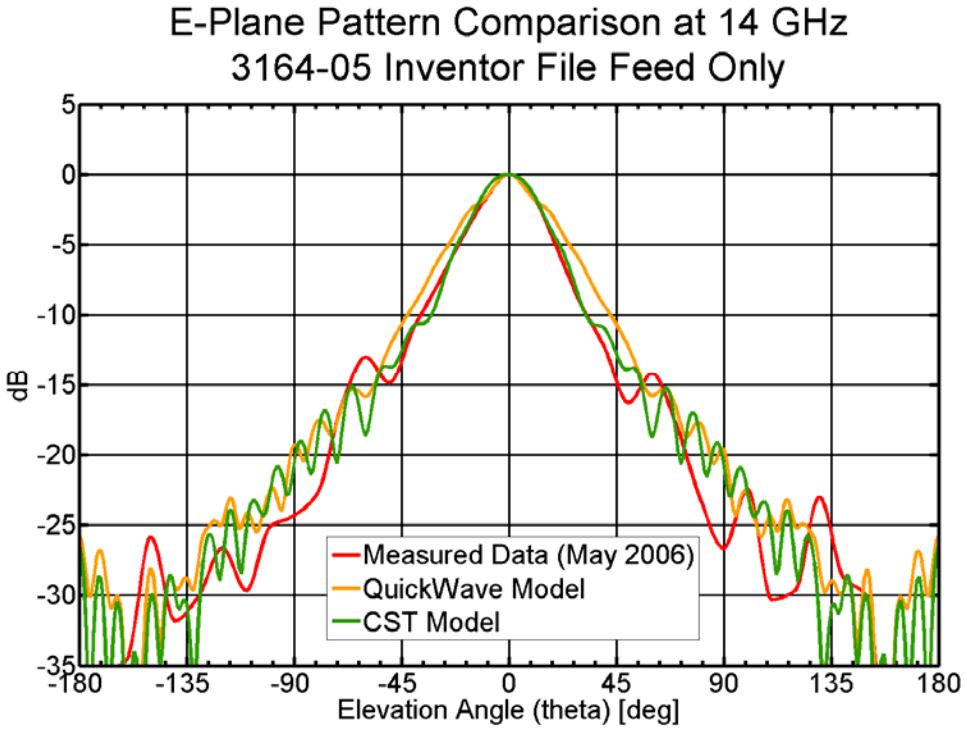


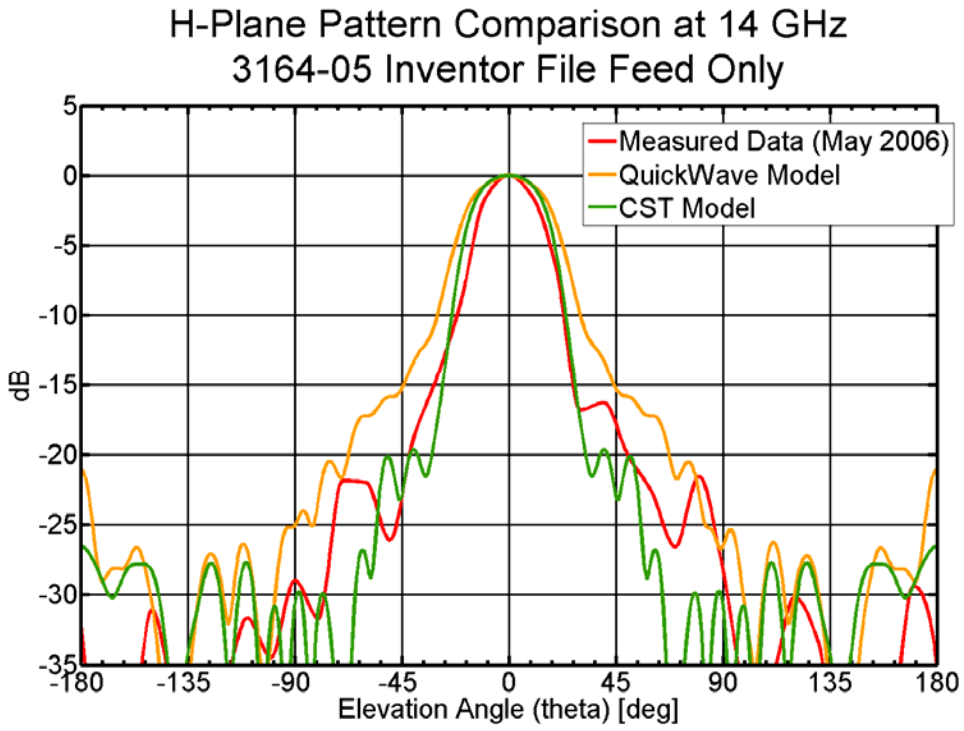
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 11 GHz:



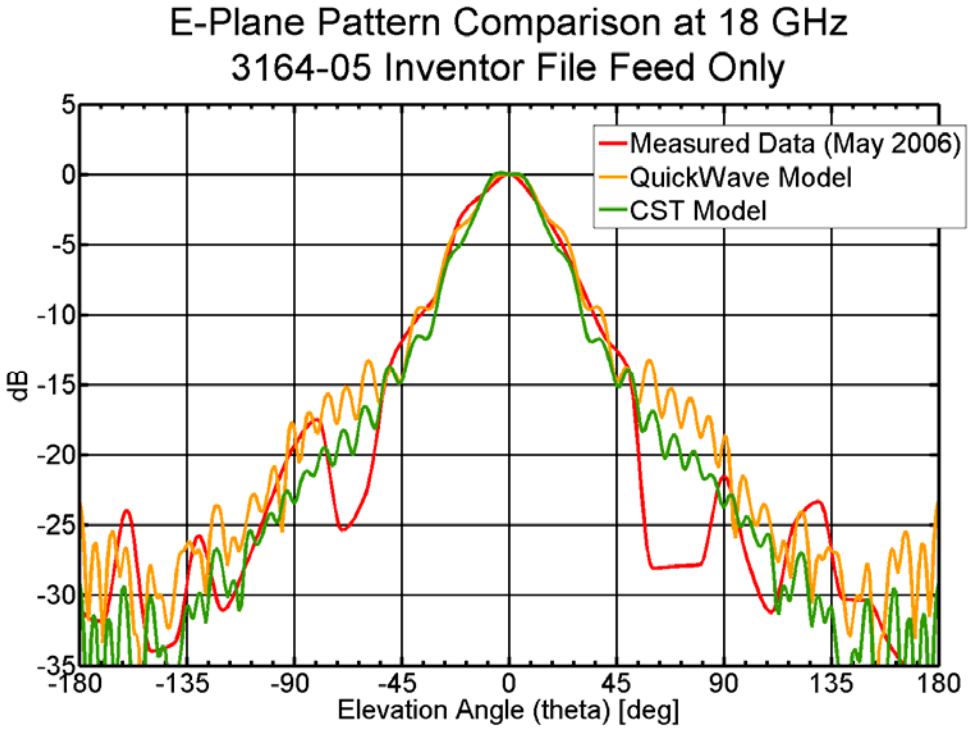


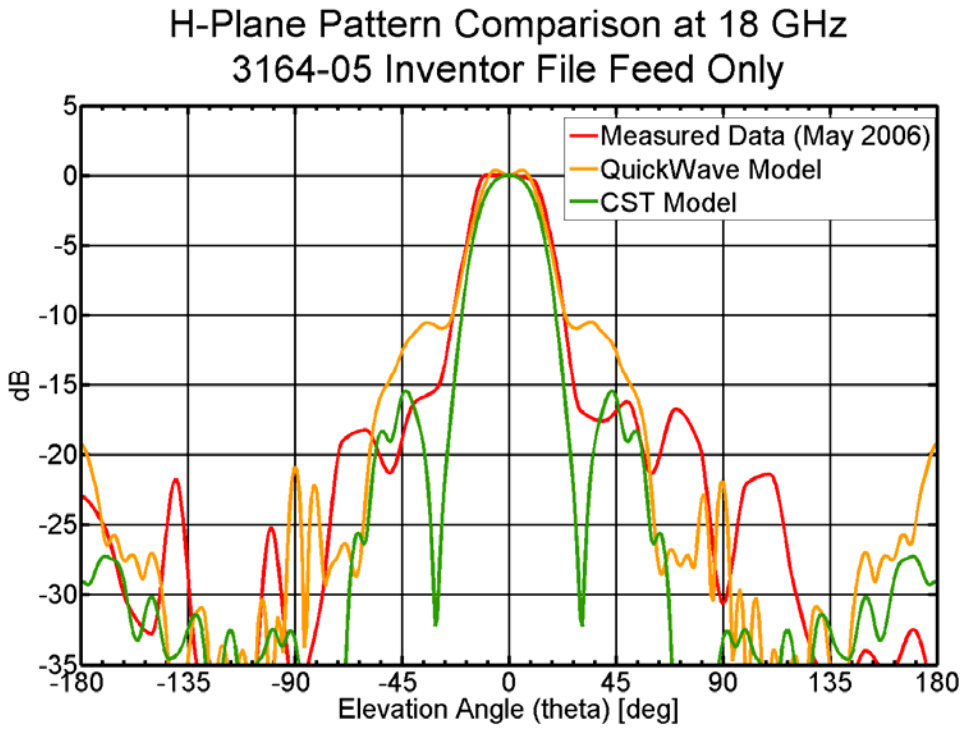
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 14 GHz:





Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 18 GHz:





6.1.3 Far-Field Patterns for Feed in Radiation Shield

The images below show the actual feed in the radiation shield and the Inventor files design model. It is clear that the fin shape is different. This is expected to result in discrepancies particularly at the low frequencies.

Lindgren 3164-05 Actual Feed

Figure 6.1.3 a

Lindgren Inventor File Design Model (CST)

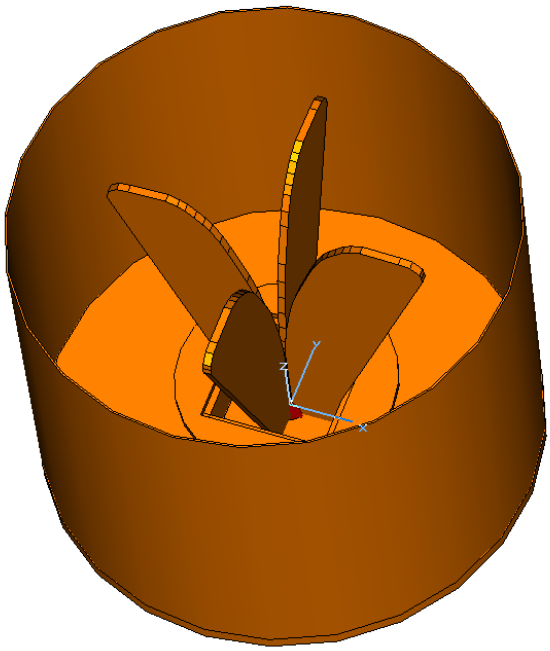
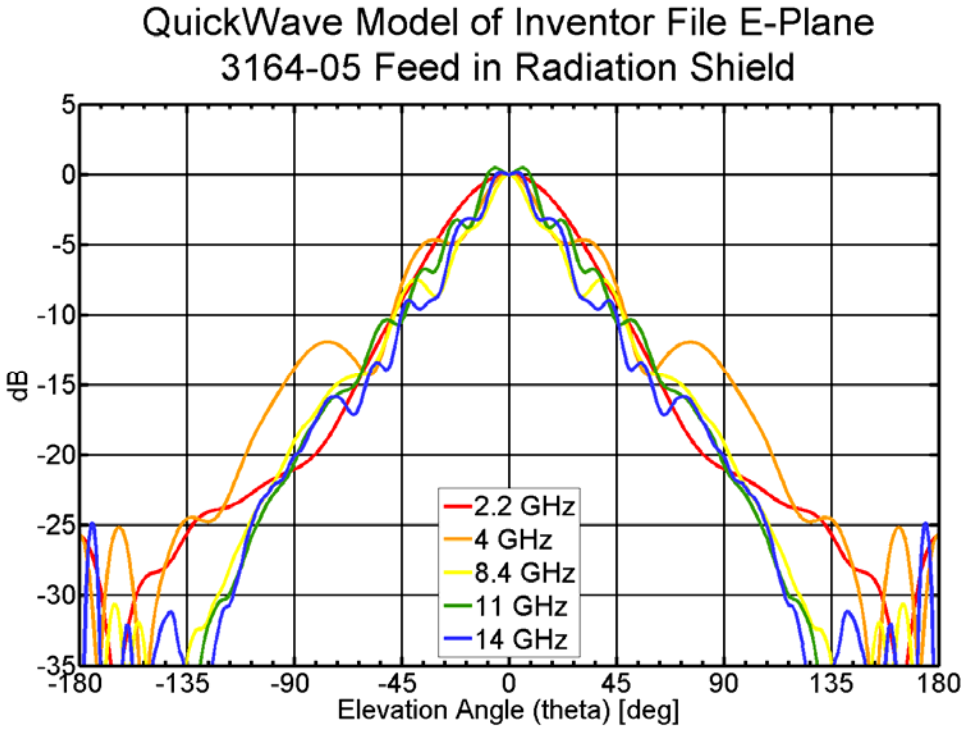
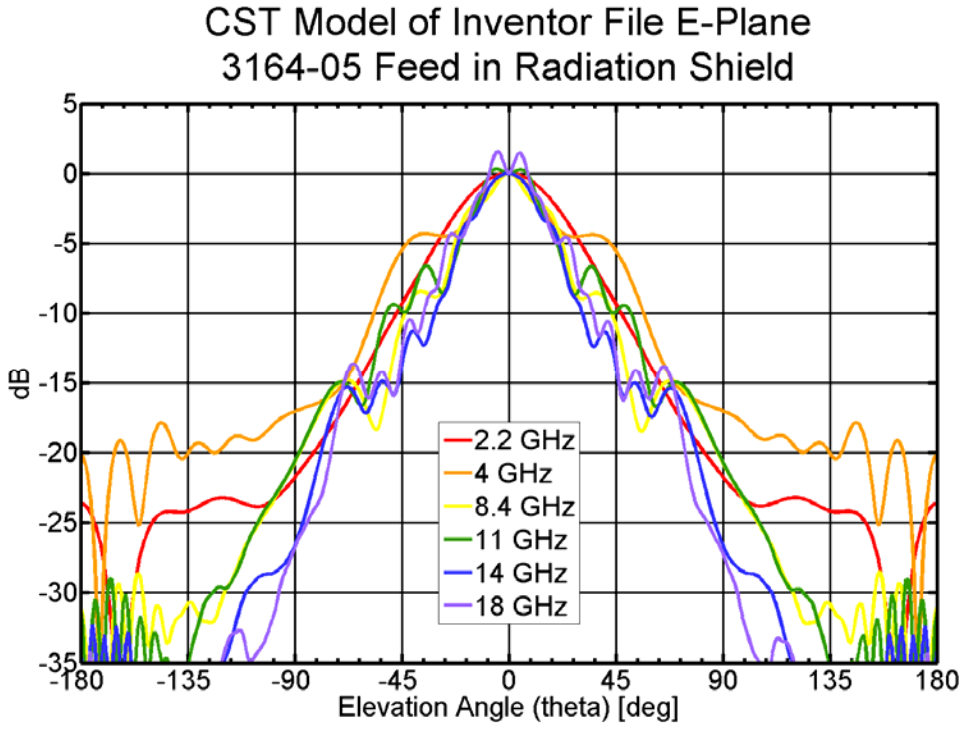


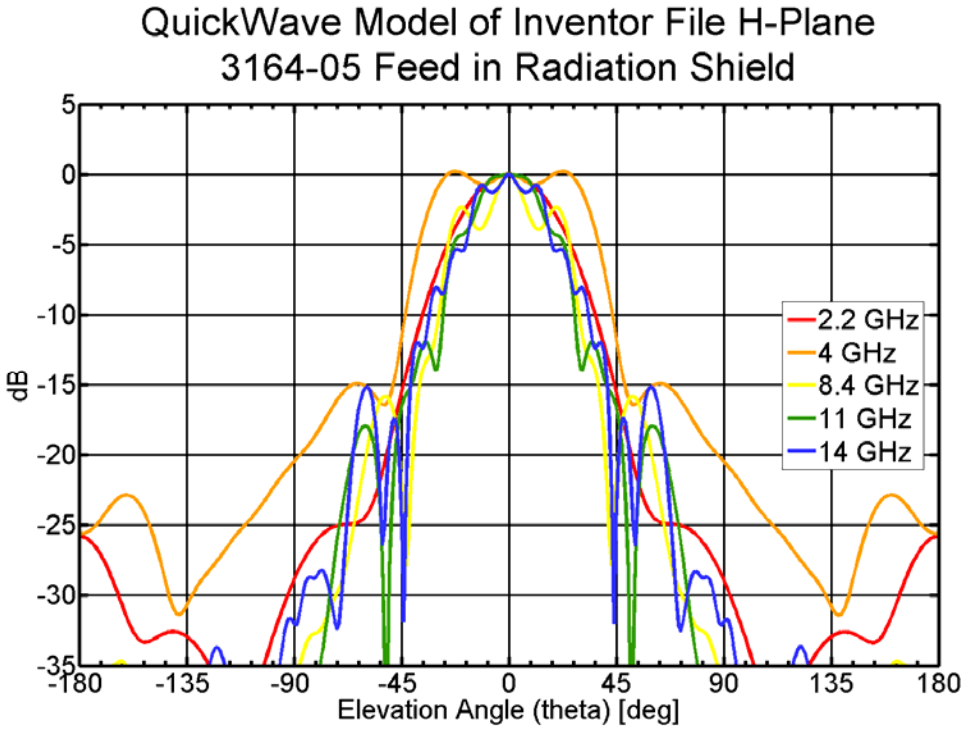
Figure 6.1.3 b

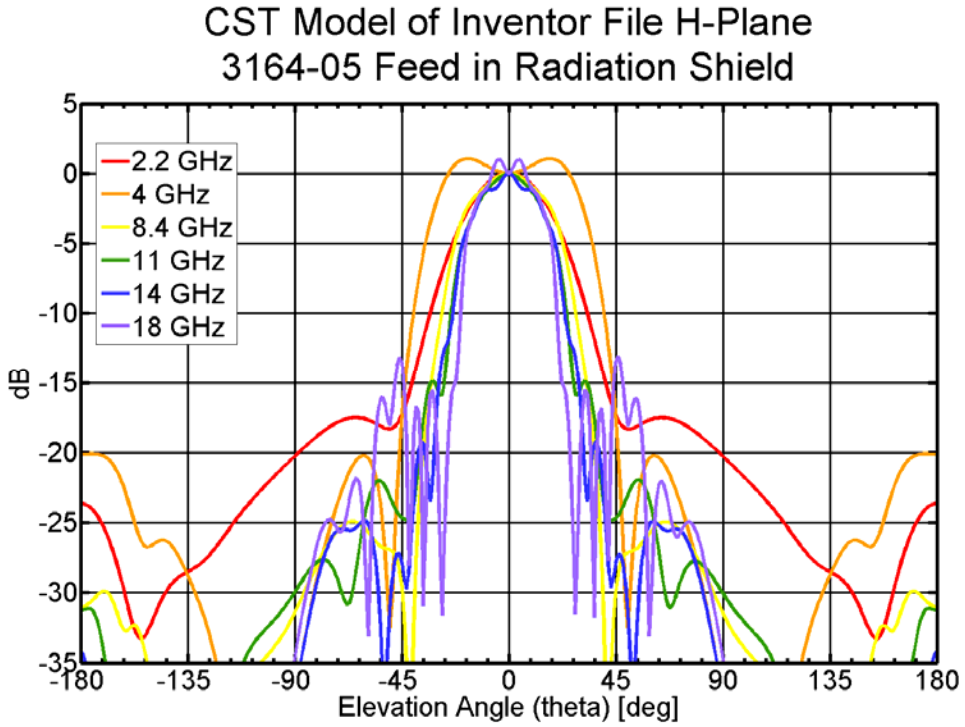
Comparison of QuickWave Model and CST Model E-Plane Patterns:



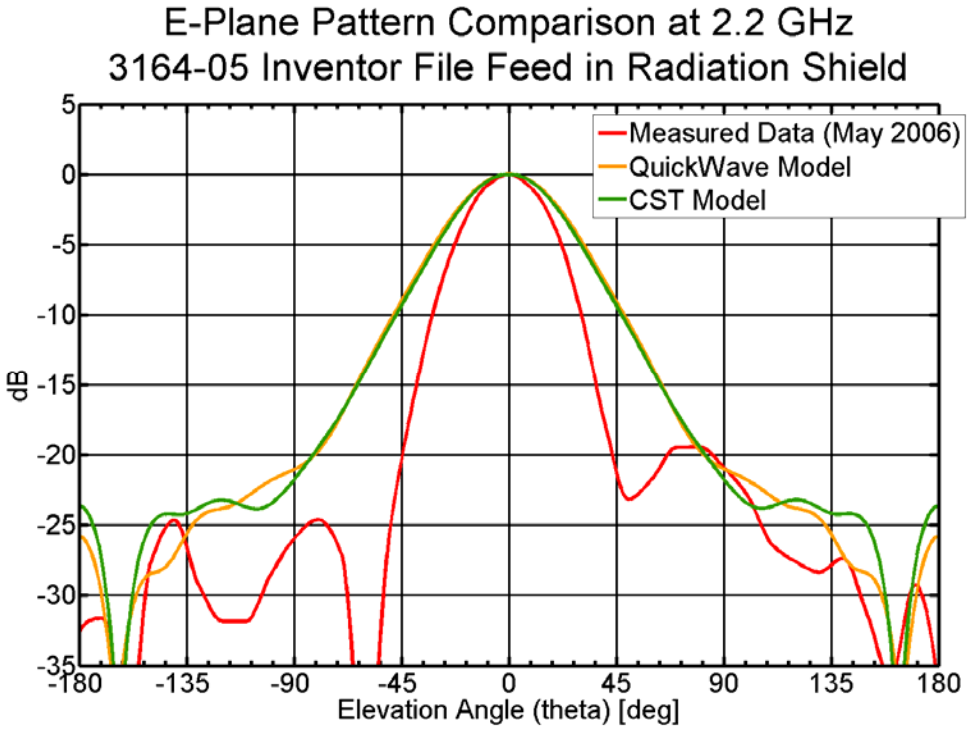


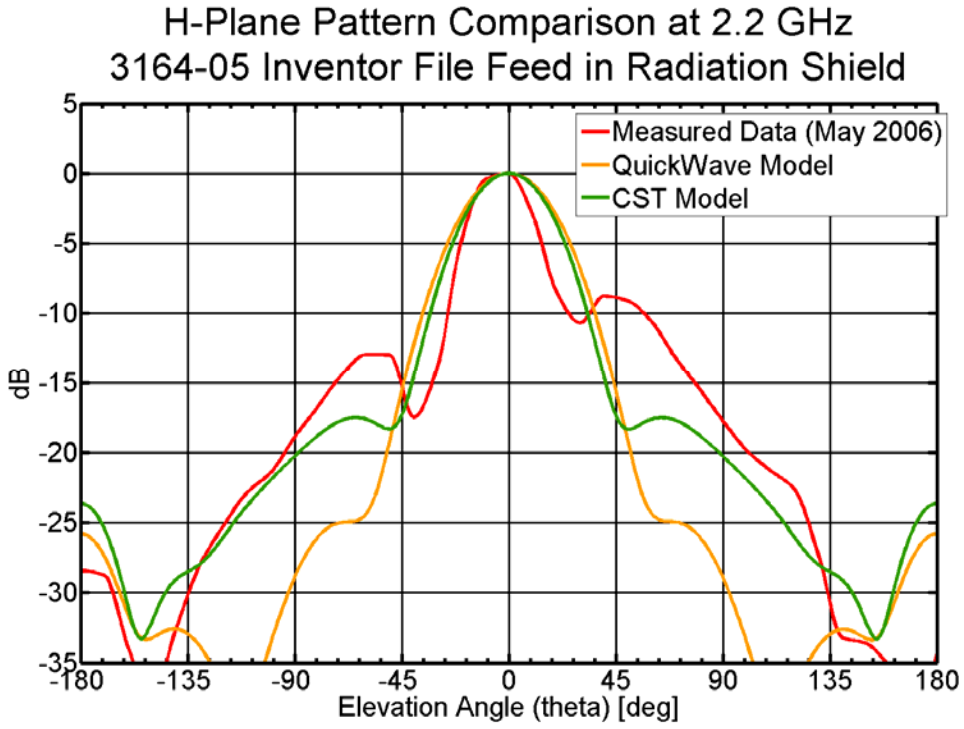
Comparison of QuickWave Model and CST Model H-Plane Patterns:



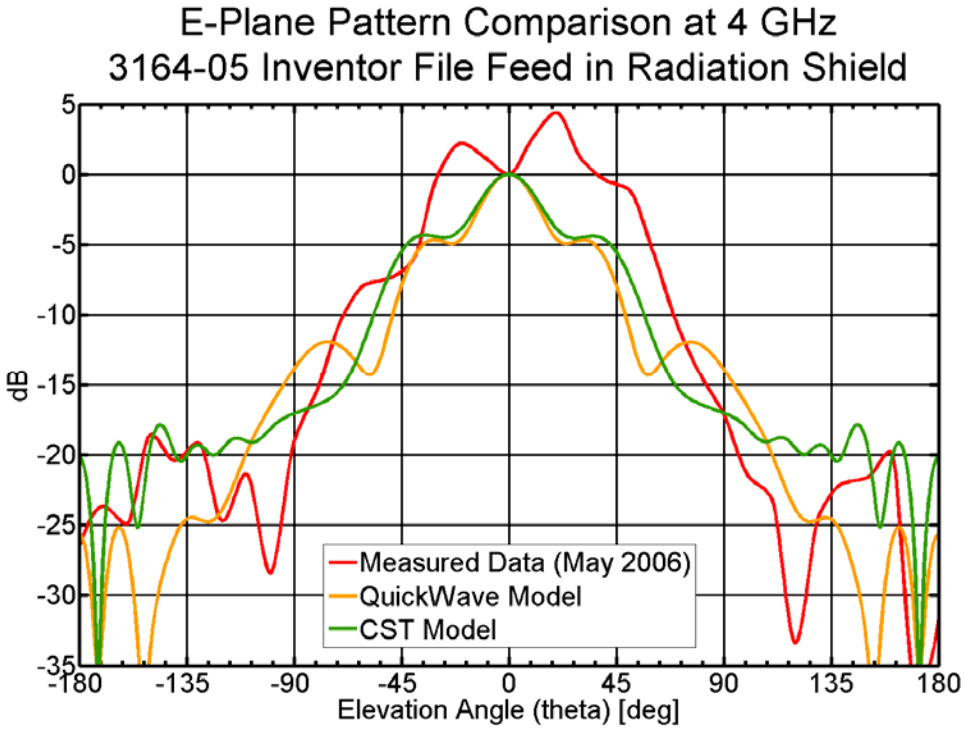


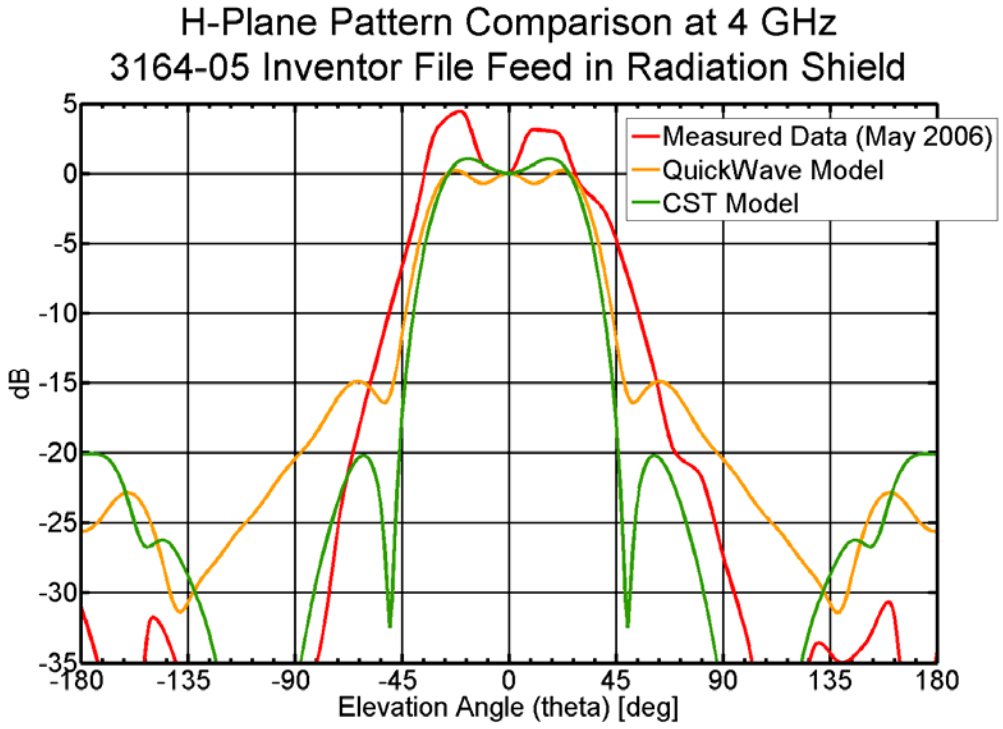
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 2.2 GHz:



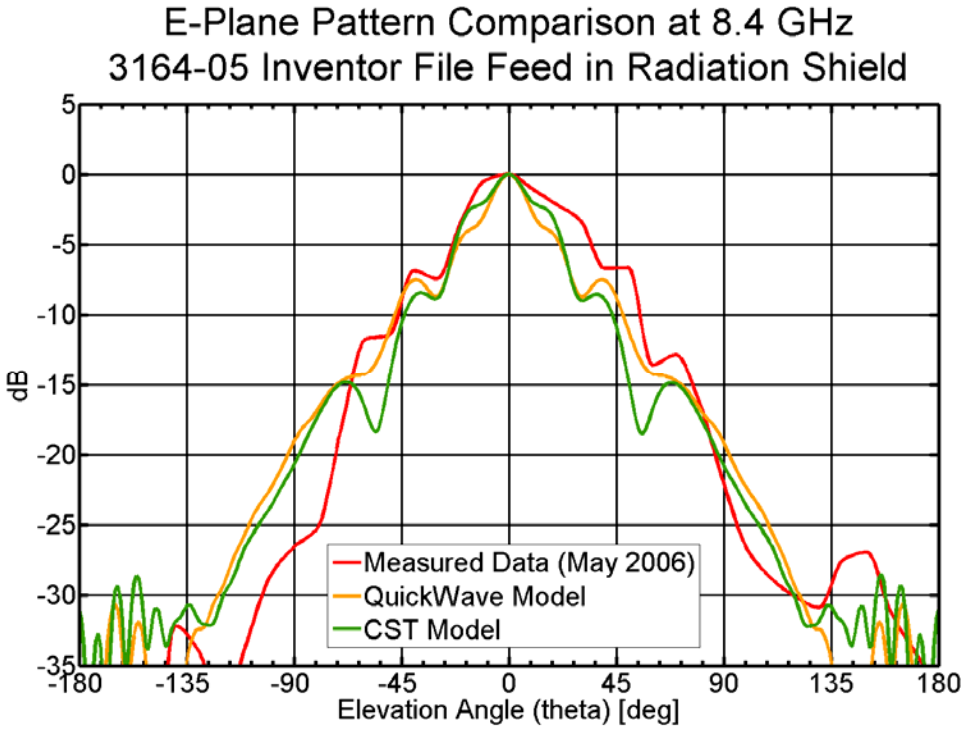


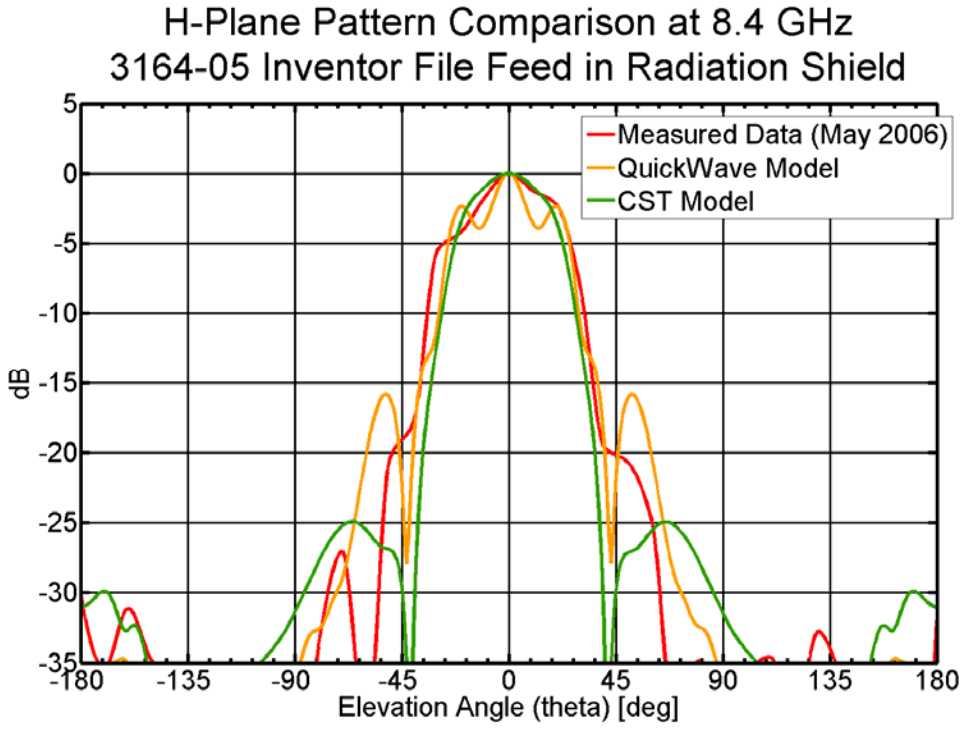
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 4.0 GHz:



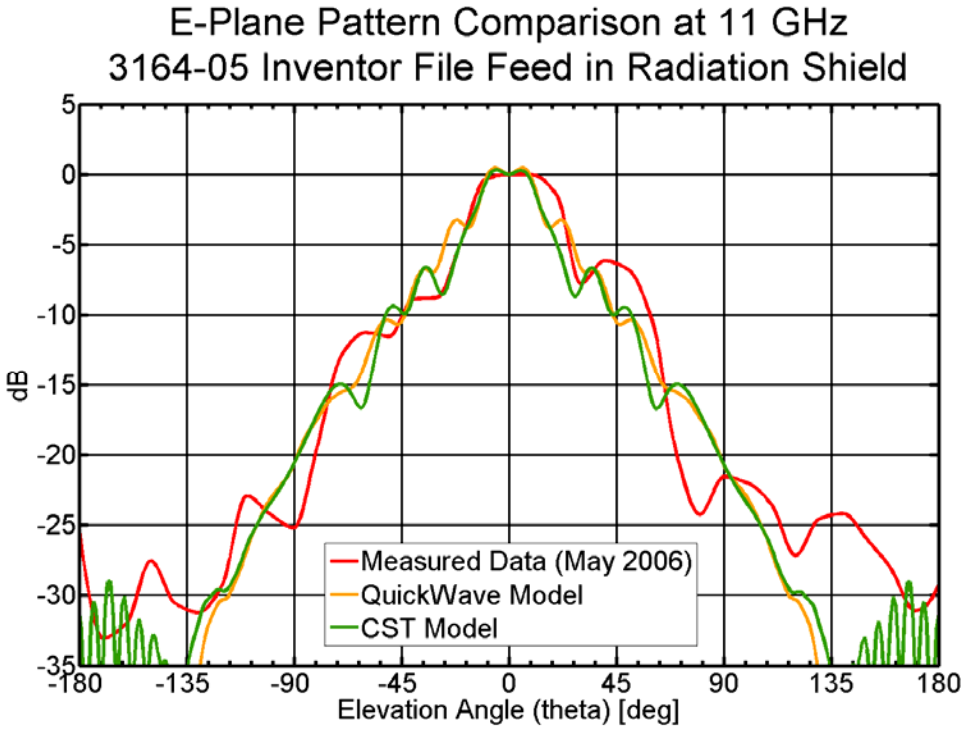


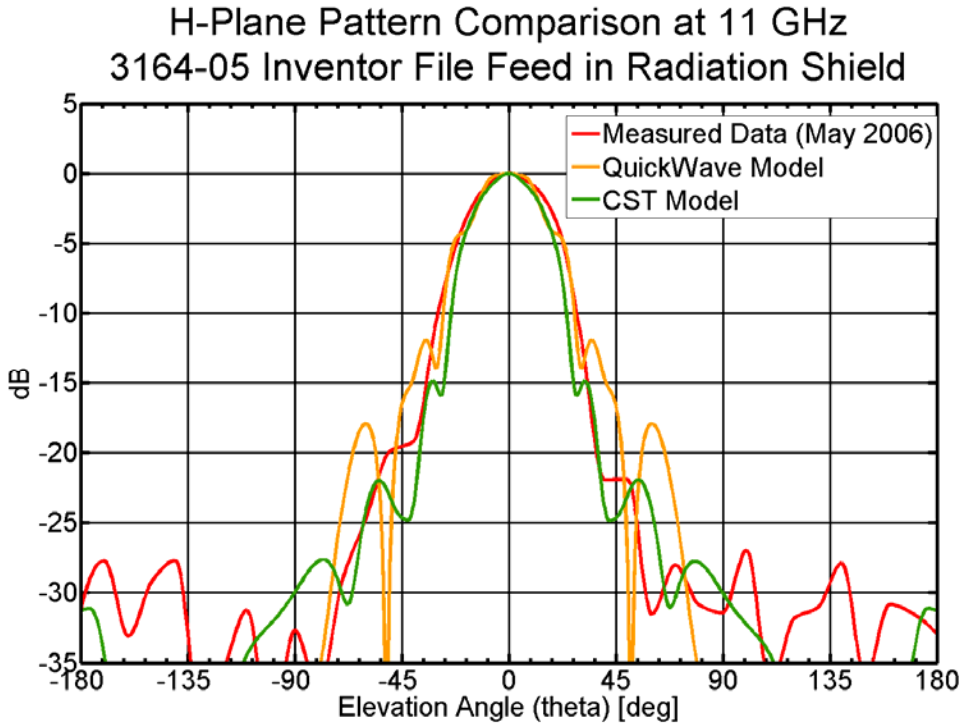
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 8.4 GHz:



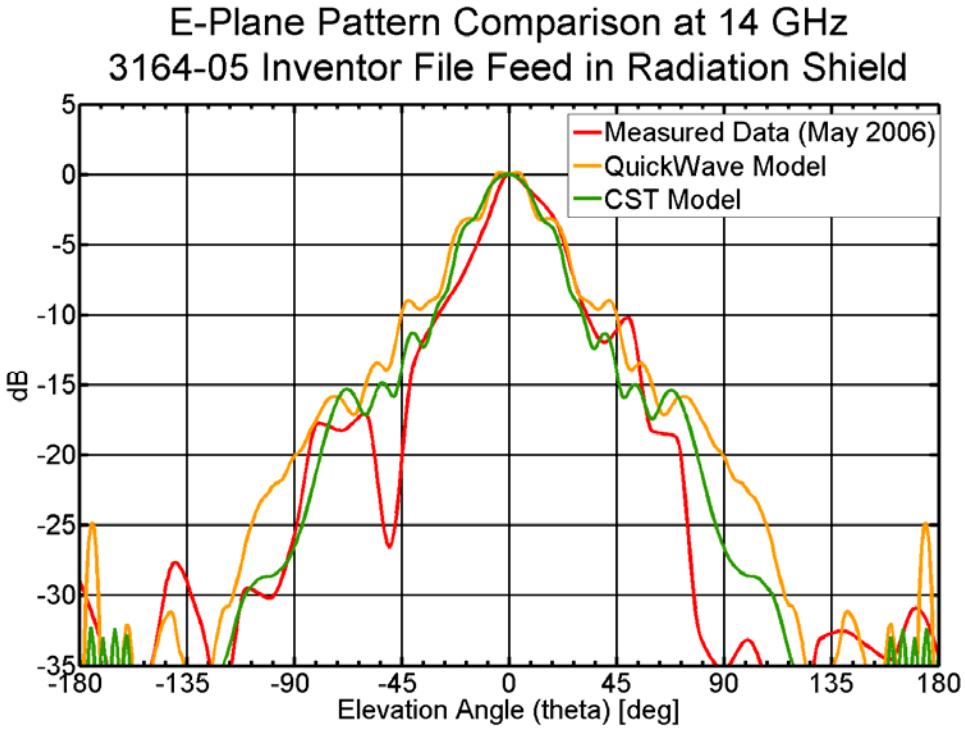


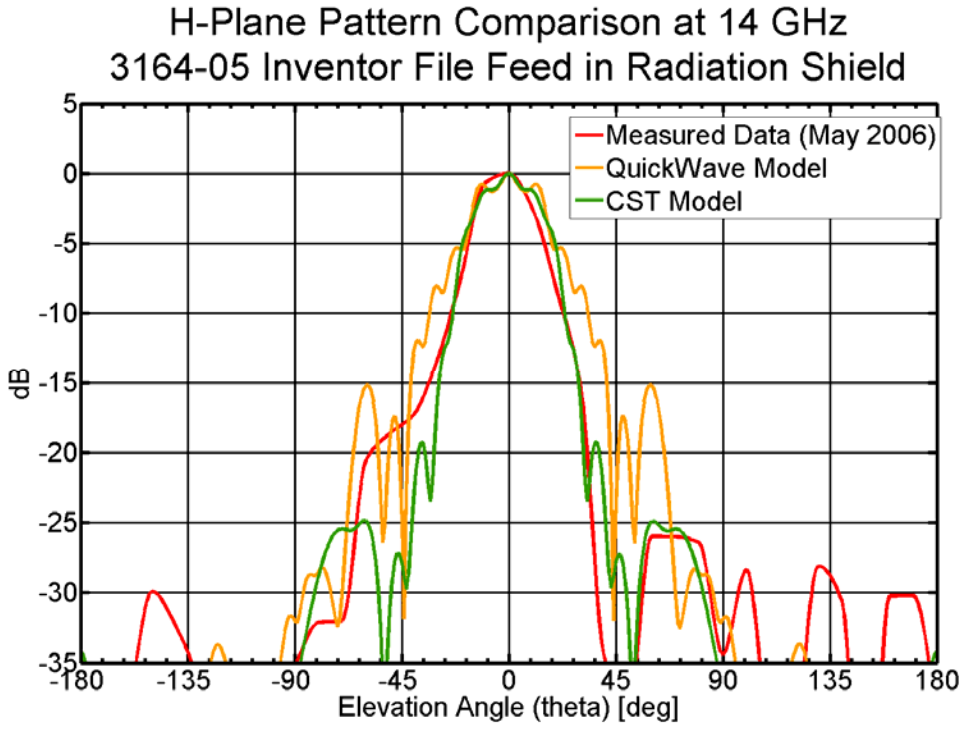
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 11 GHz:



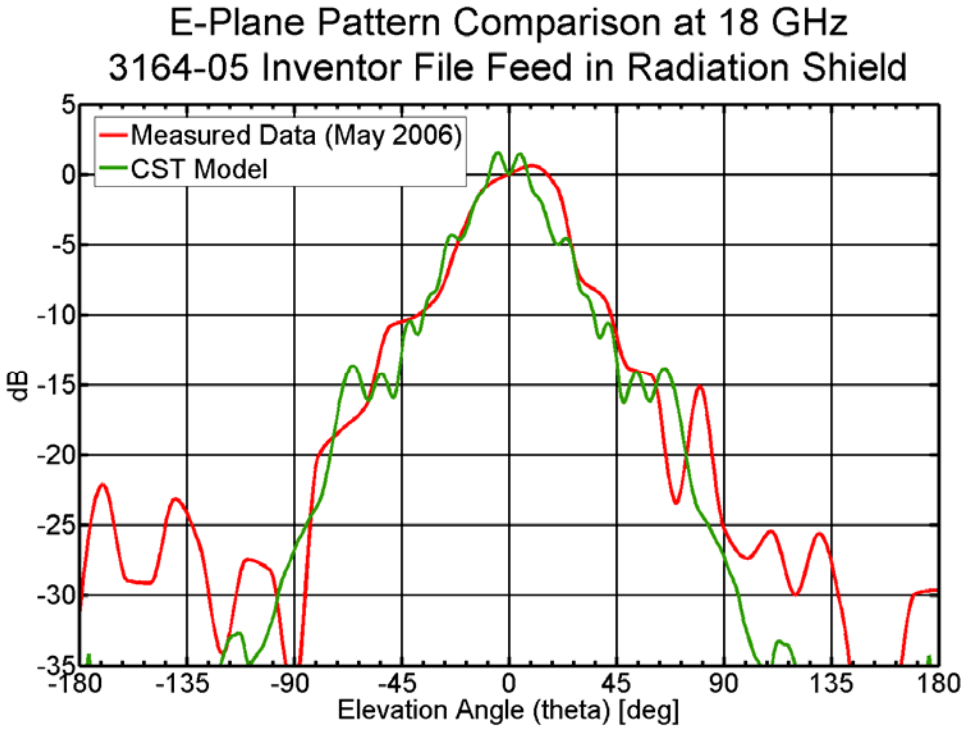


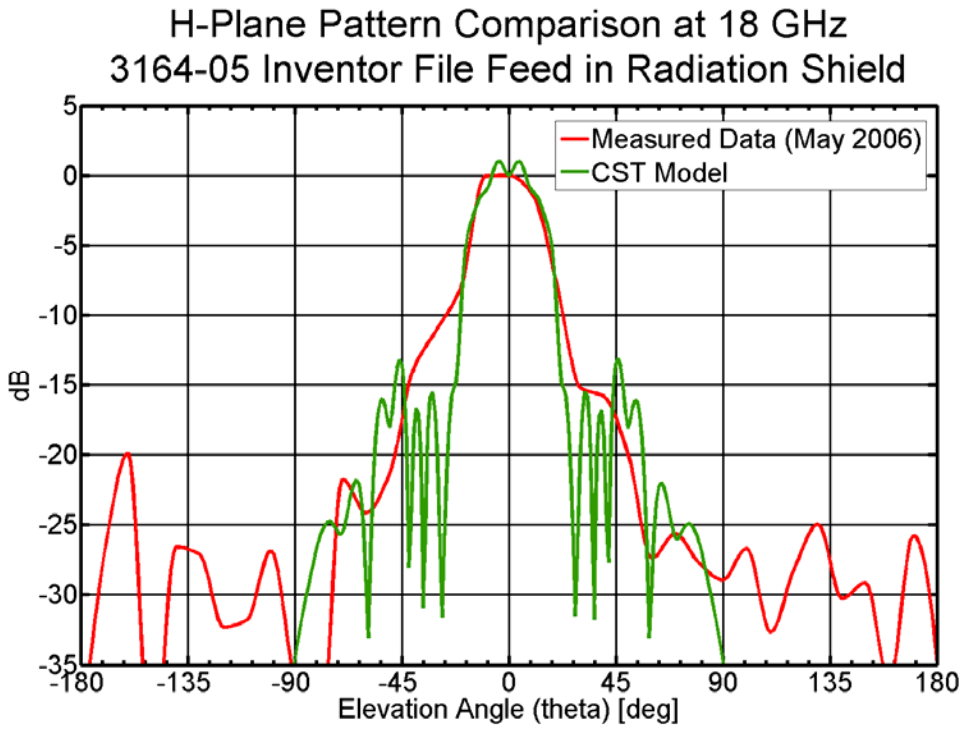
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 14 GHz:





Comparison of Measured Data and CST Model Patterns at 18 GHz:





6.1.4 Far-Field Patterns for Feed in Absorber Lined Radiation Shield

The images below show the actual feed in the AN-73 absorber lined radiation shield and the Inventor files design model. It is clear that the fin shape is different. This is expected to result in discrepancies particularly at the low frequencies.

Lindgren 3164-05 Actual Feed

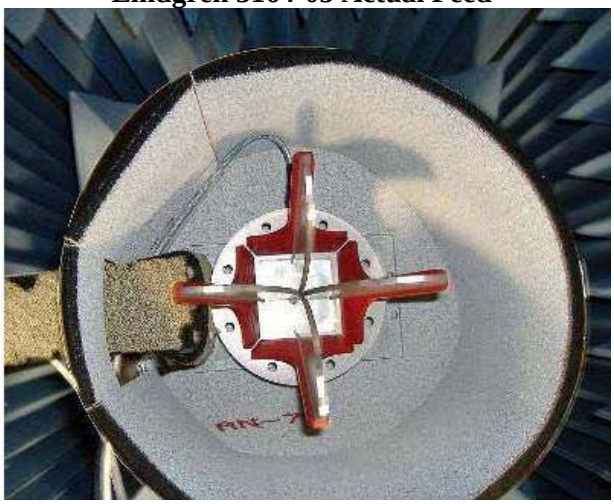


Figure 6.1.4 a

Lindgren Inventor File Design Model (CST)

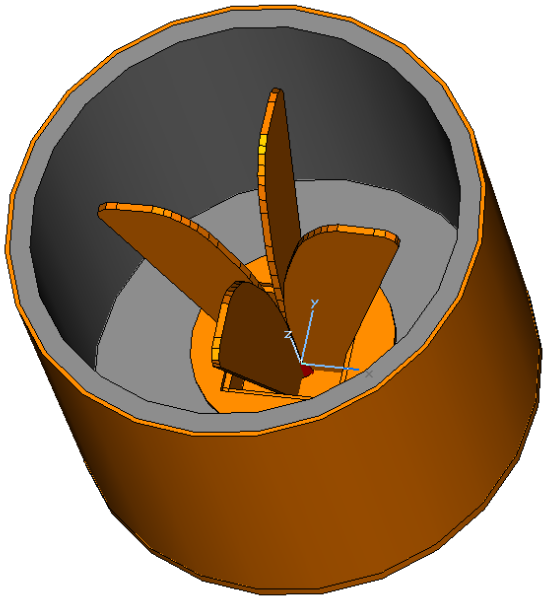
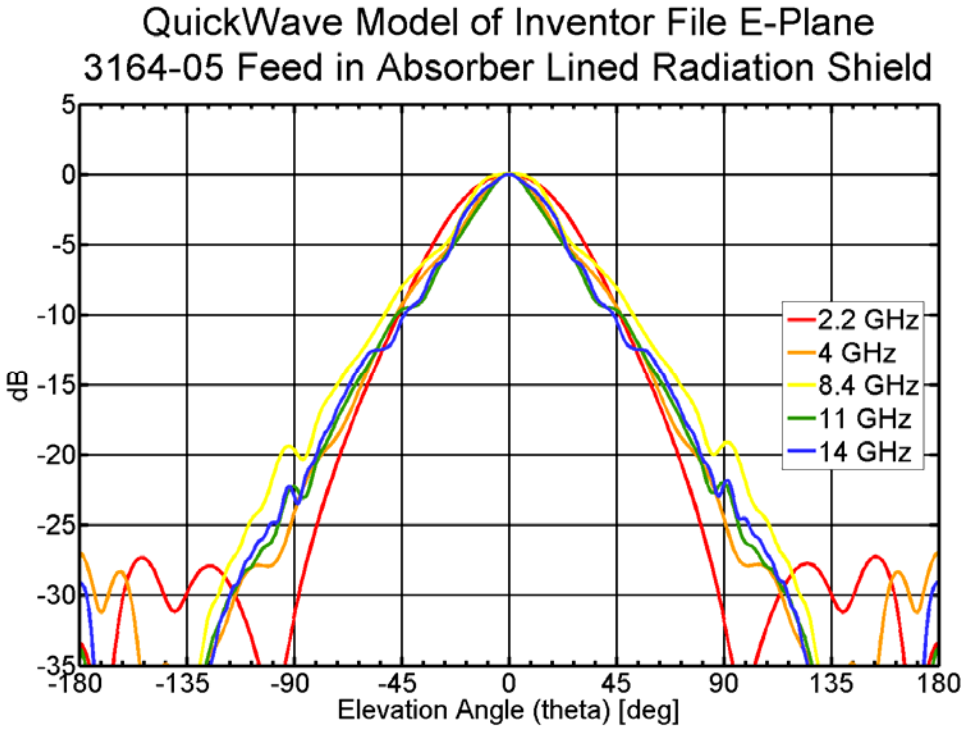
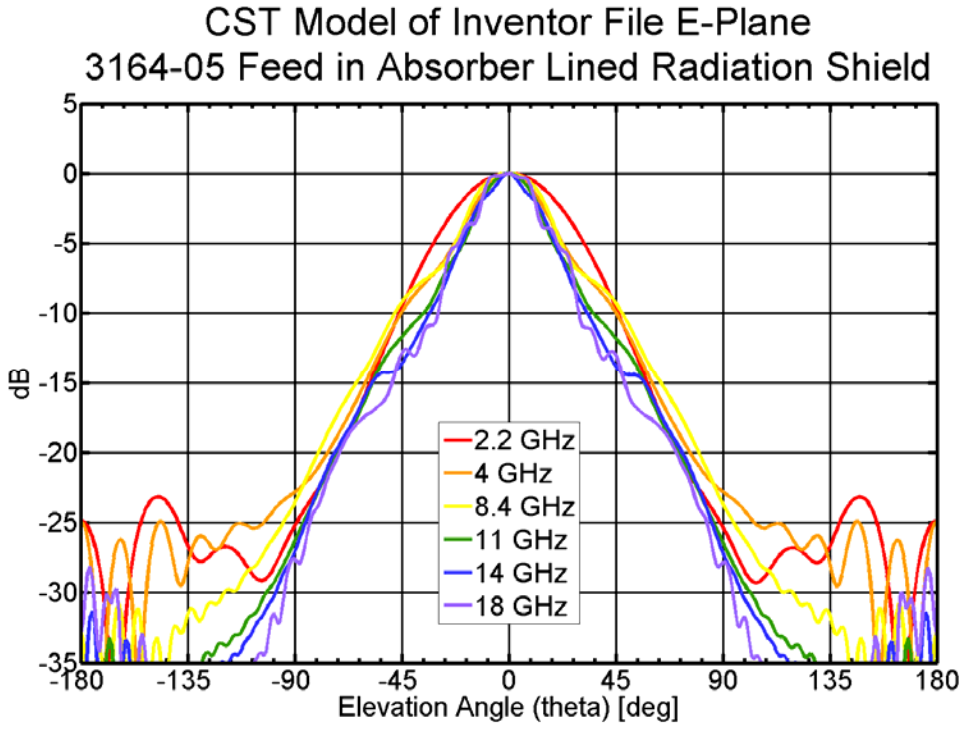


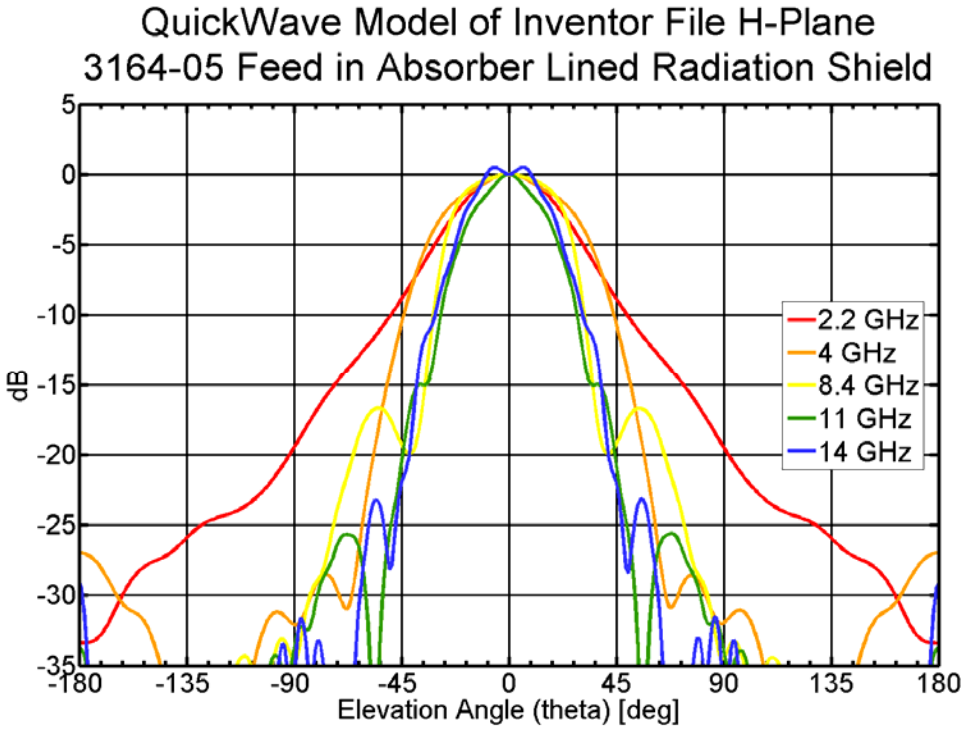
Figure 6.1.4 b

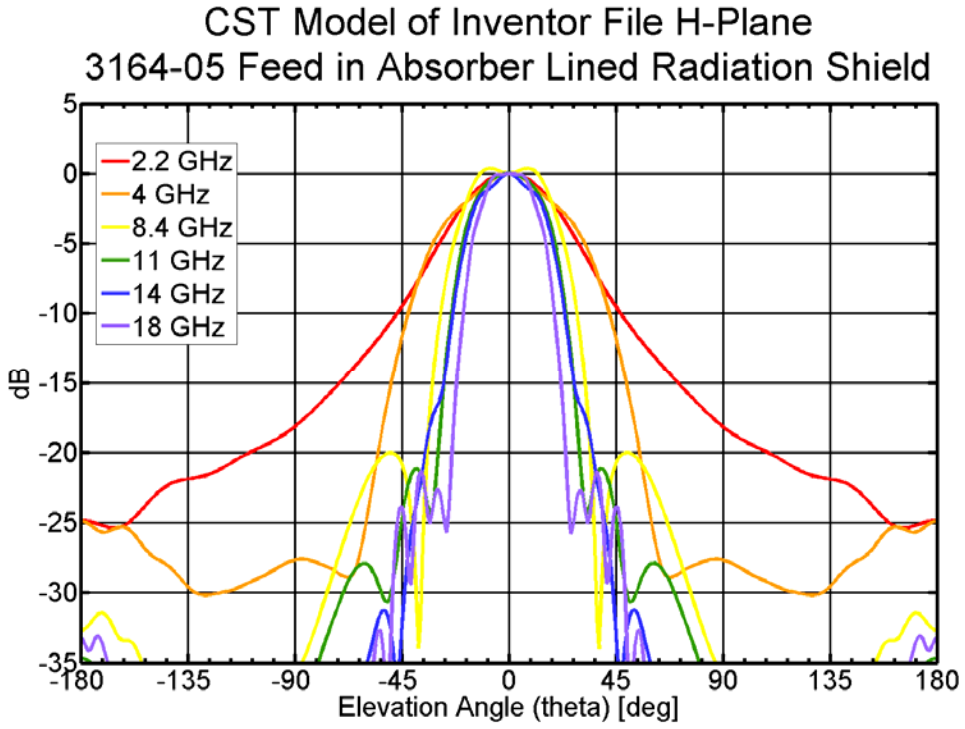
Comparison of QuickWave Model and CST Model E-Plane Patterns:



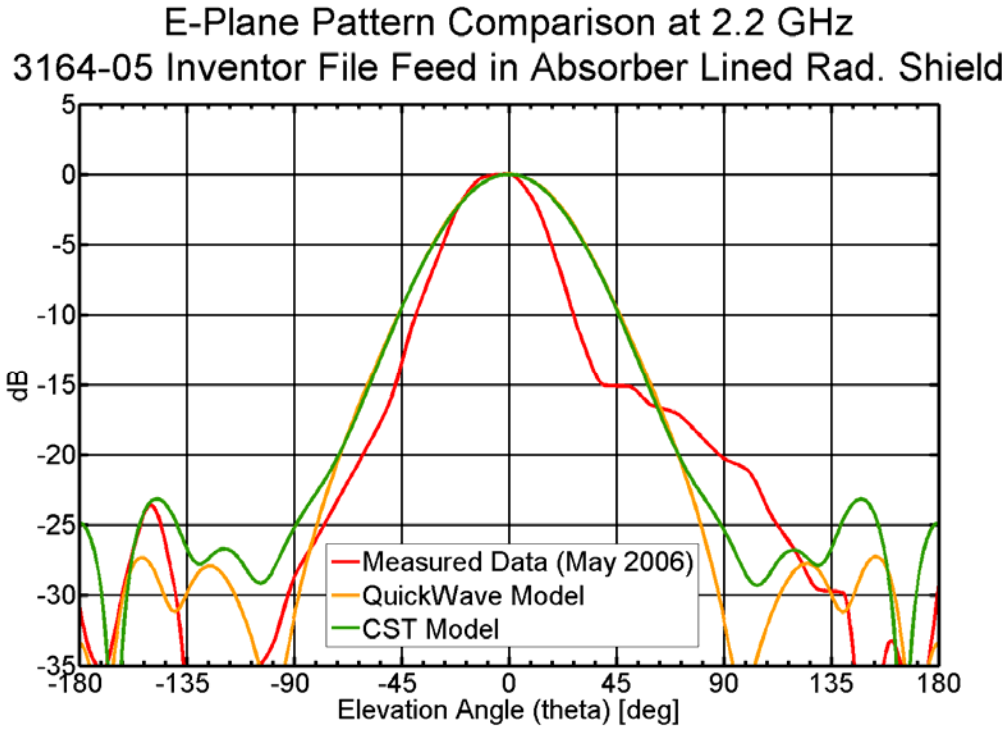


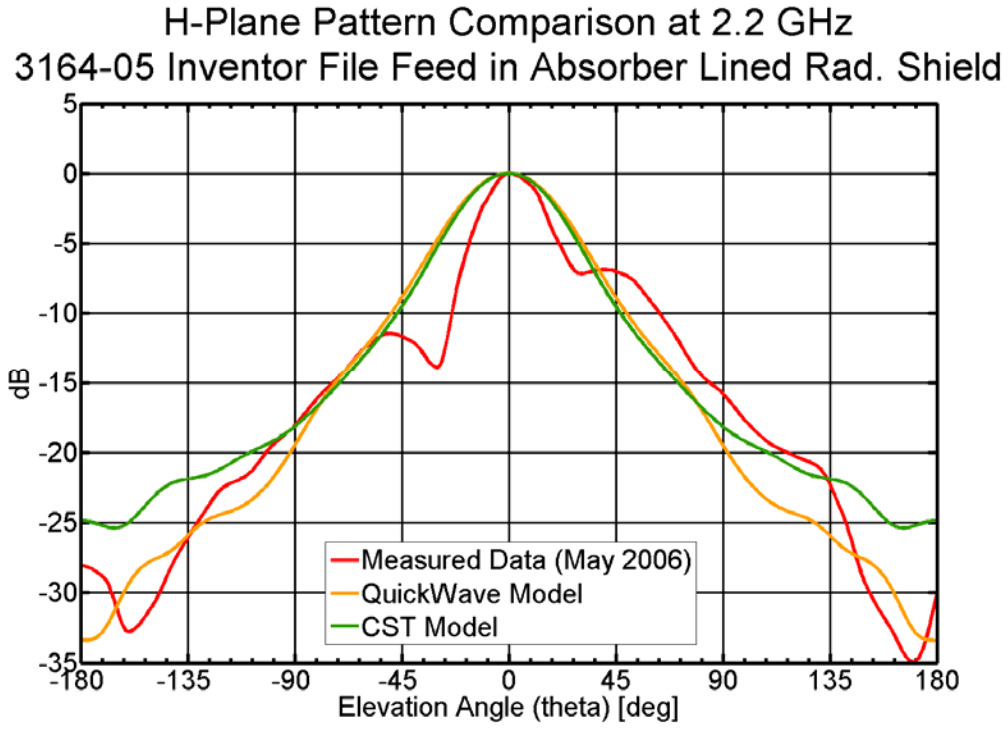
Comparison of QuickWave Model and CST Model H-Plane Patterns:



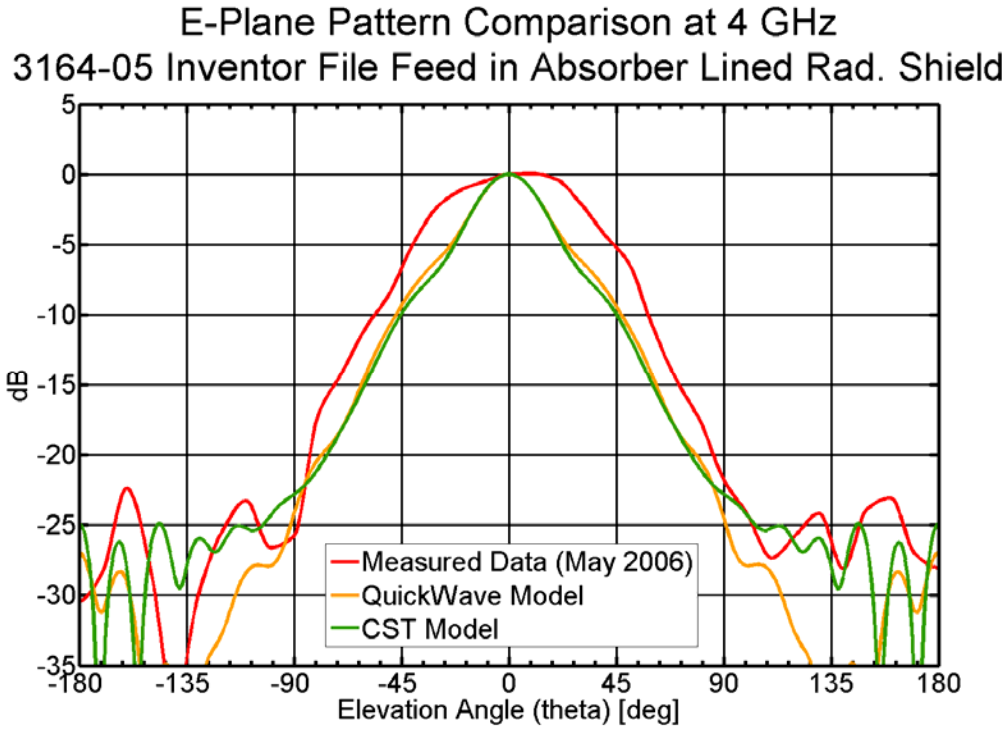


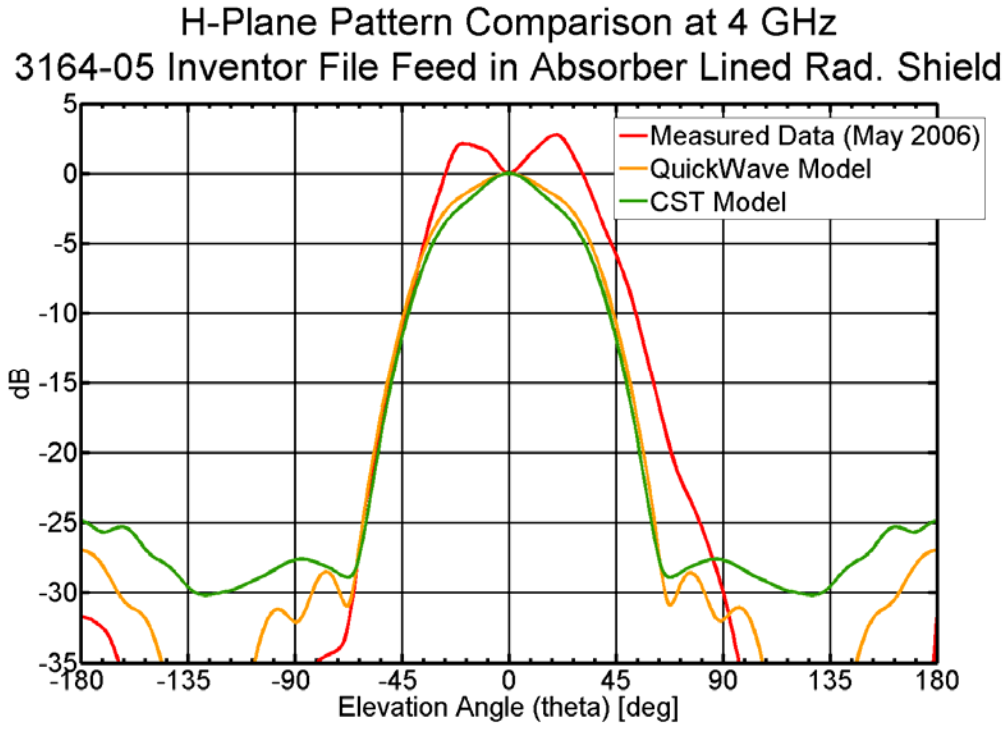
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 2.2 GHz:



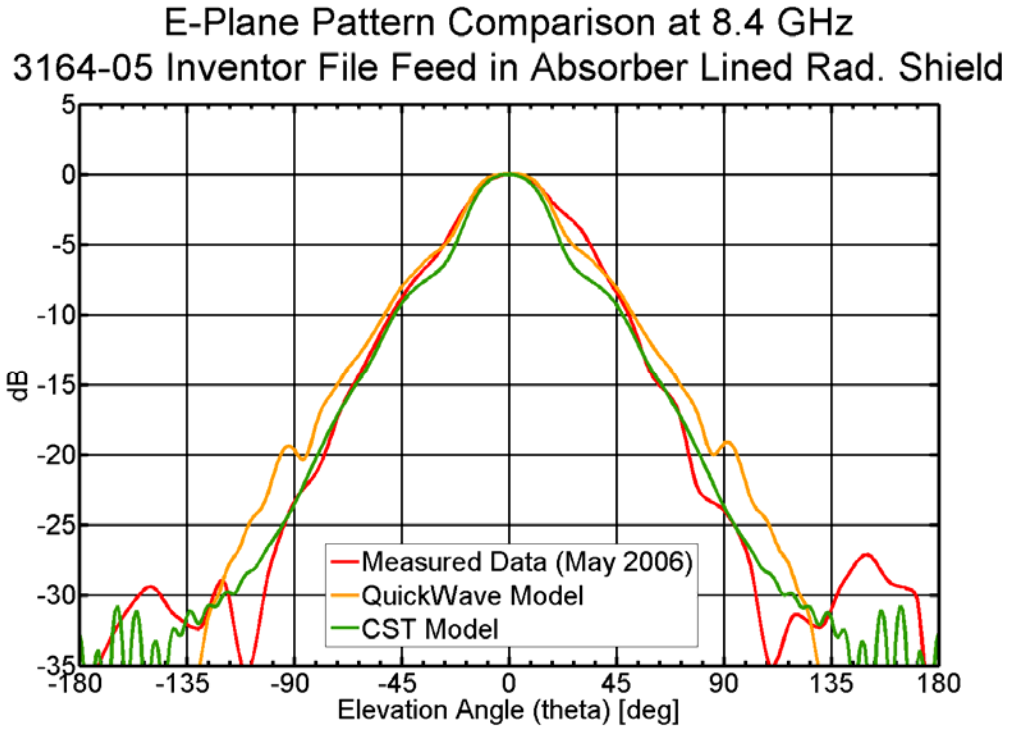


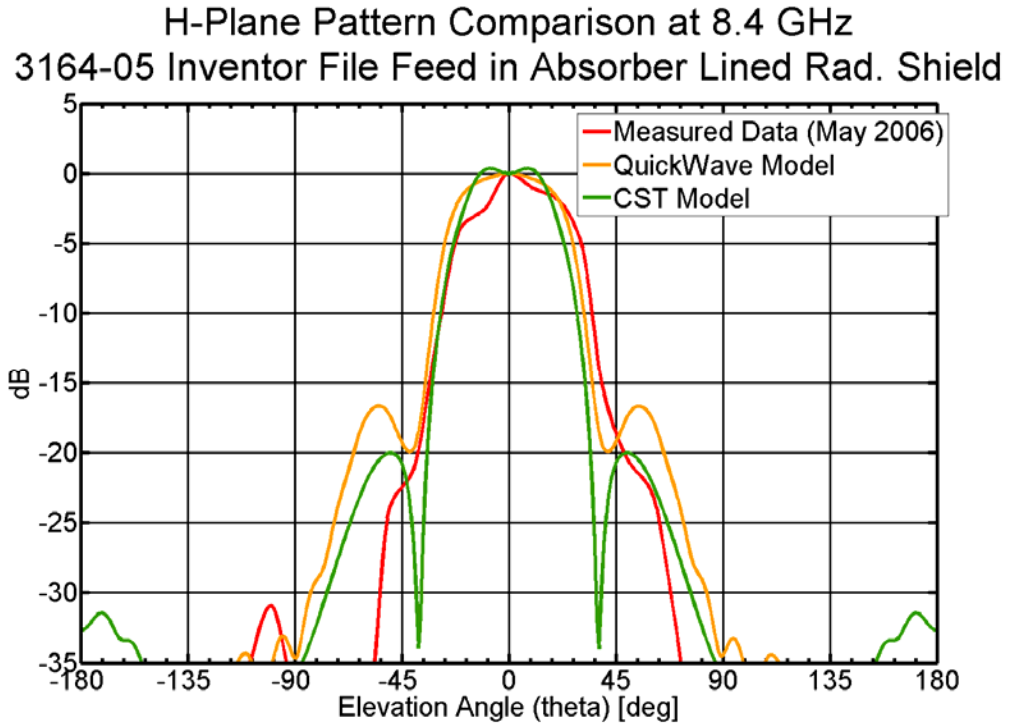
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 4.0 GHz:



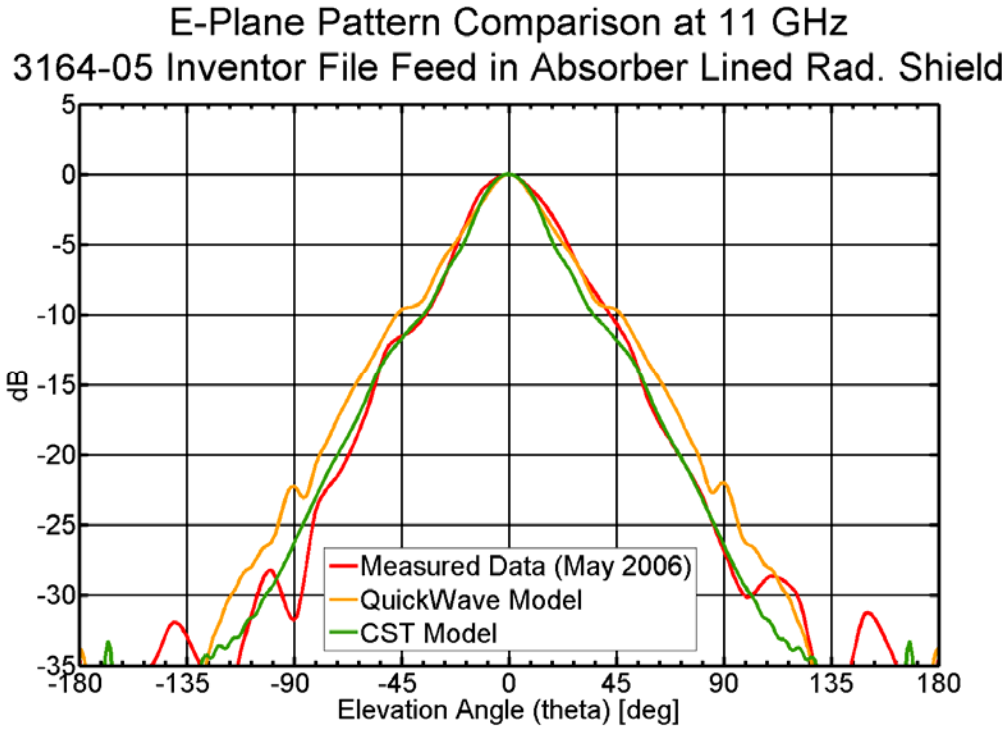


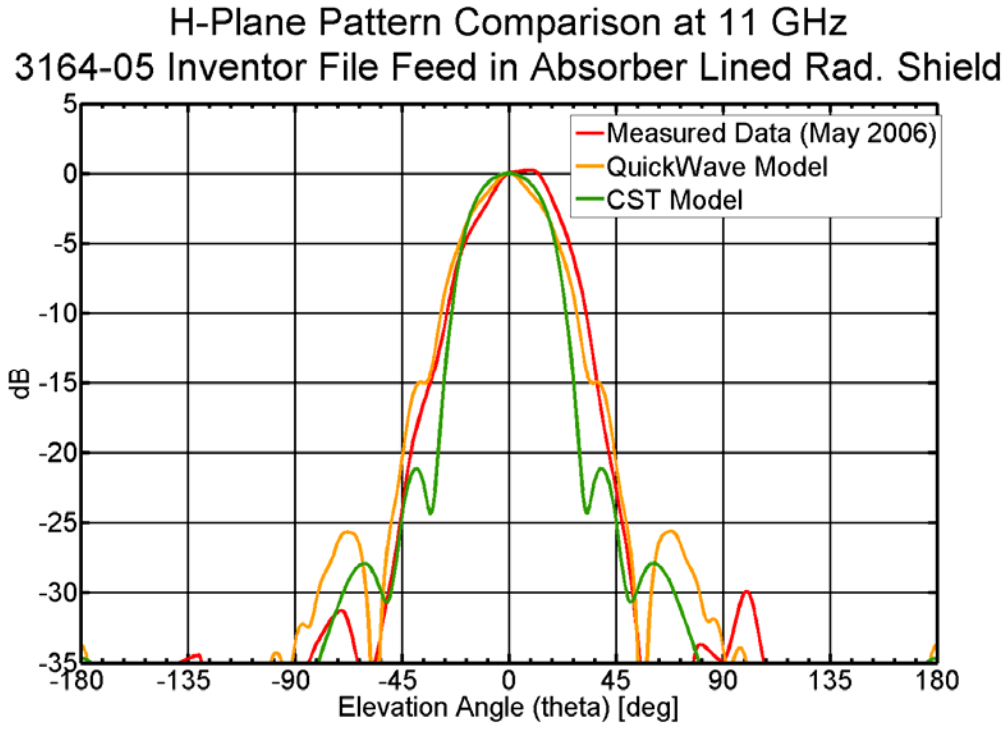
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 8.4 GHz:



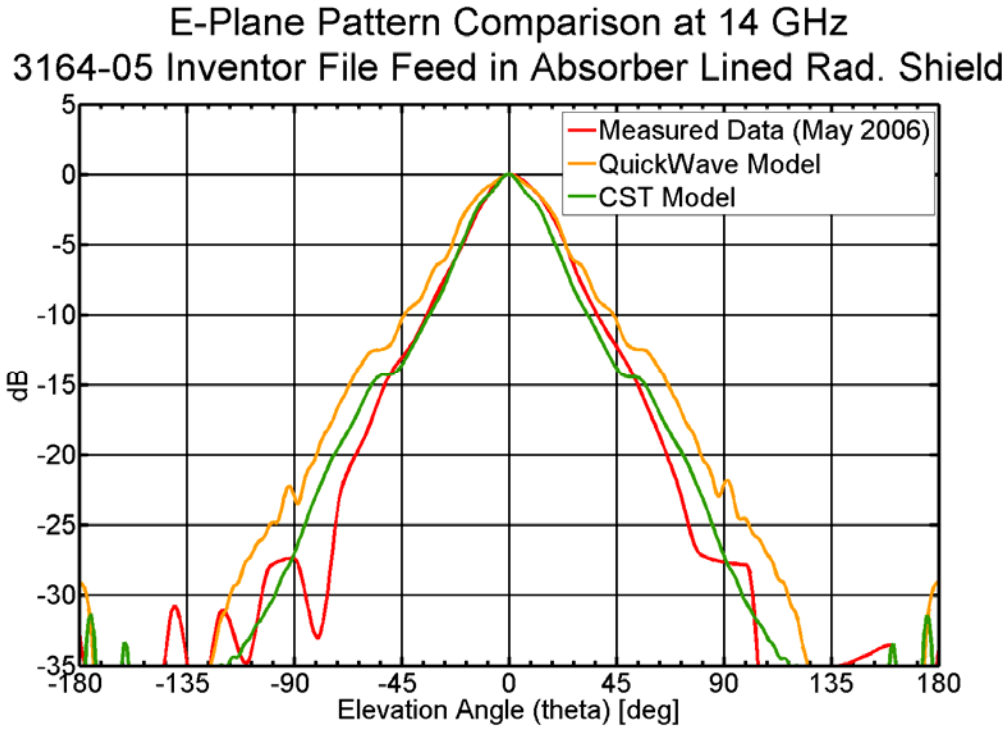


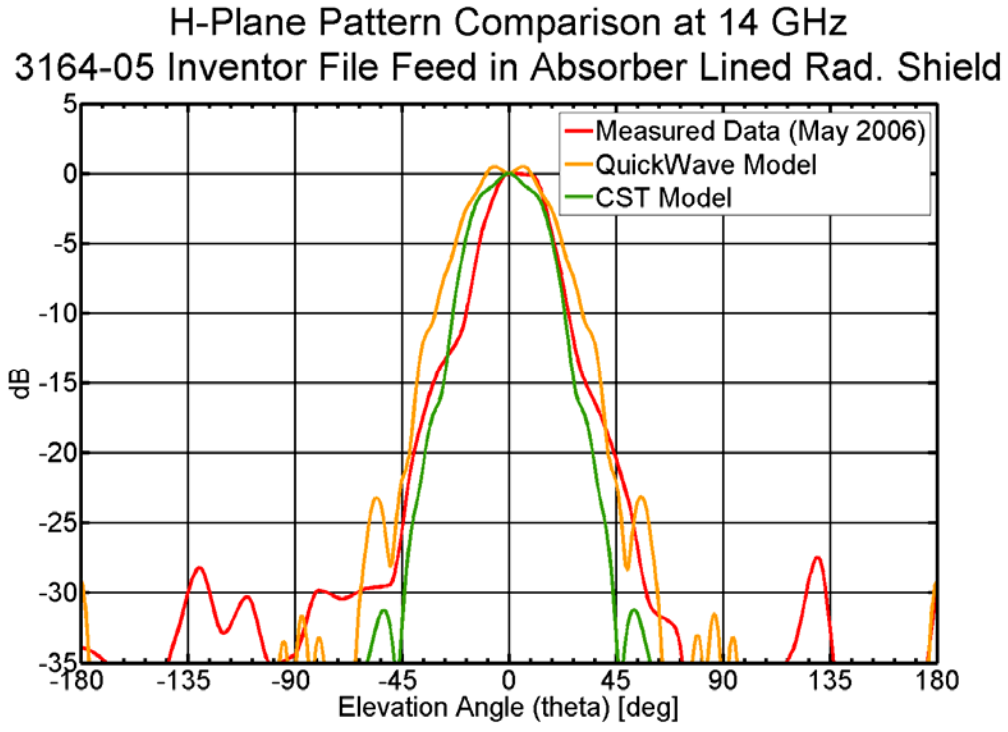
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 11 GHz:



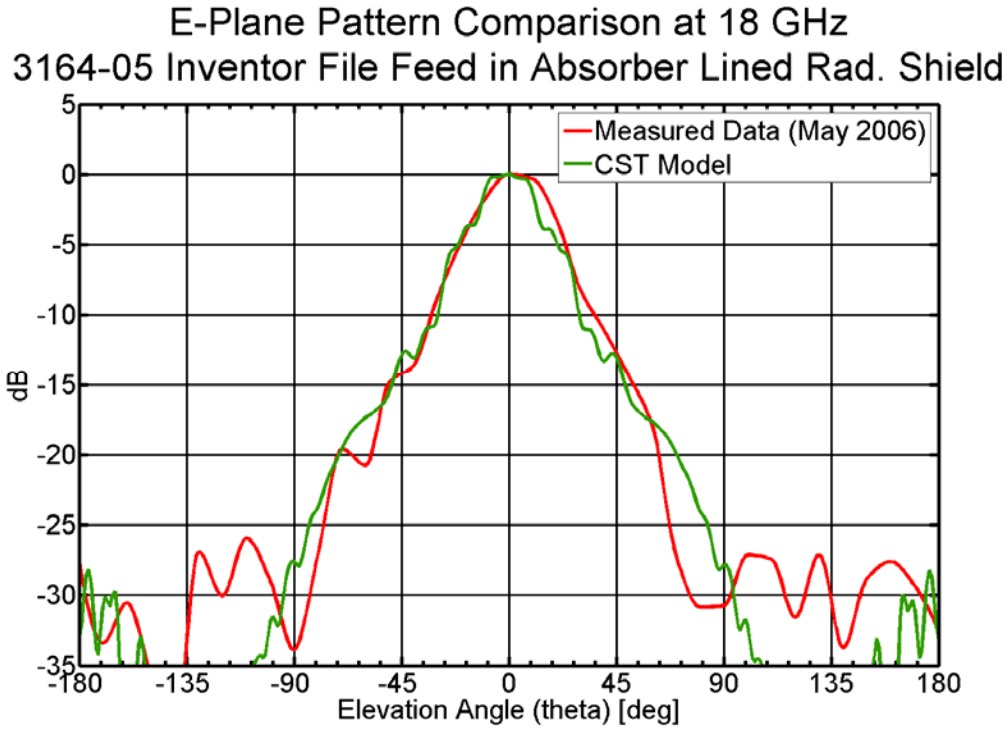


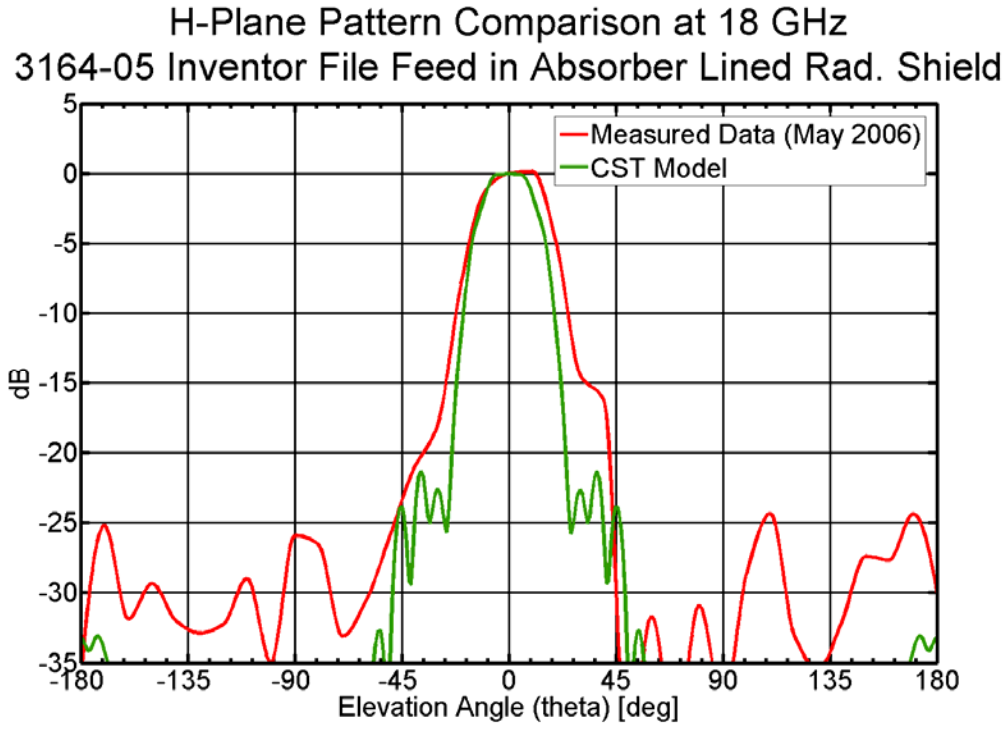
Comparison of Measured Data, QuickWave Model, and CST Model Patterns at 14 GHz:





Comparison of Measured Data and CST Model Patterns at 18 GHz:



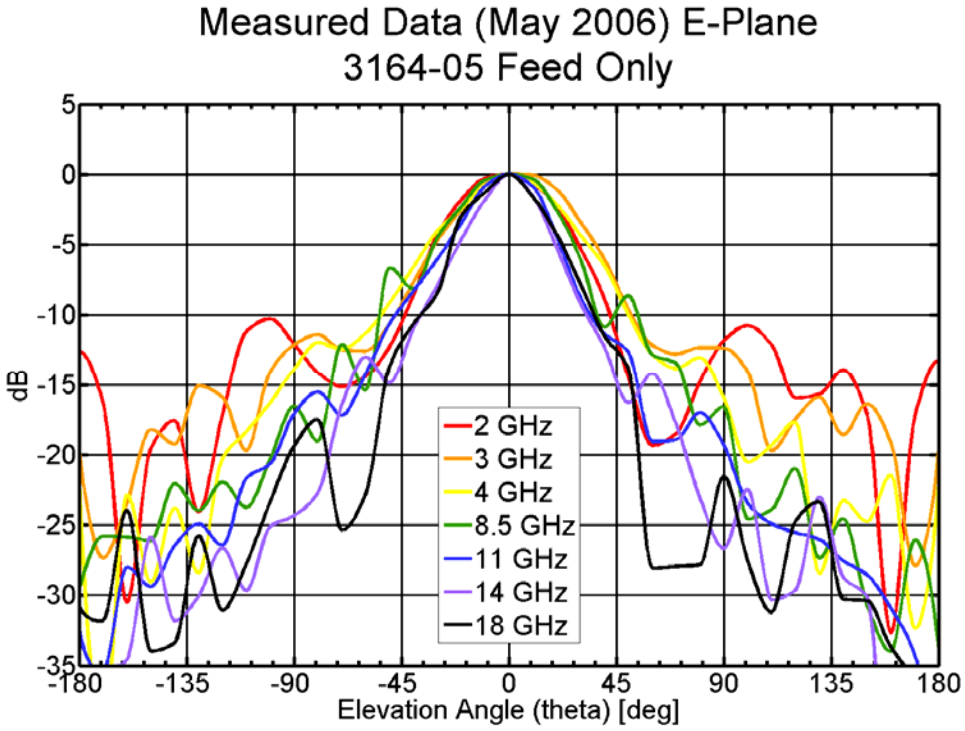


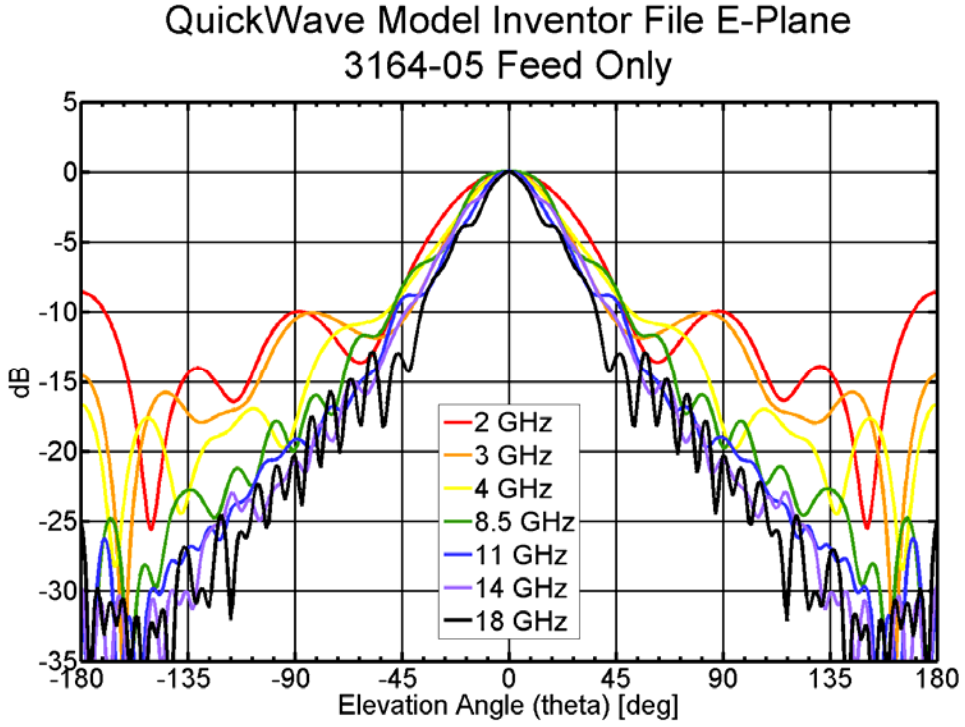
6.2 Data from QuickWave Model of Inventor File Design (1-18 GHz)

The Inventor file design of the ETS-Lindgren 3164-05 (see section 3.1.1) was modeled and simulated using QuickWave. The following plots show the results for this model. This is the same model that is compared to measured data and the CST model in section 6.1. It is important to realize that the Inventor file design is different from the actual feed, which the measured data was taken from. The shape of the fins are different. This is expected to result in discrepancies particularly at the low frequencies.

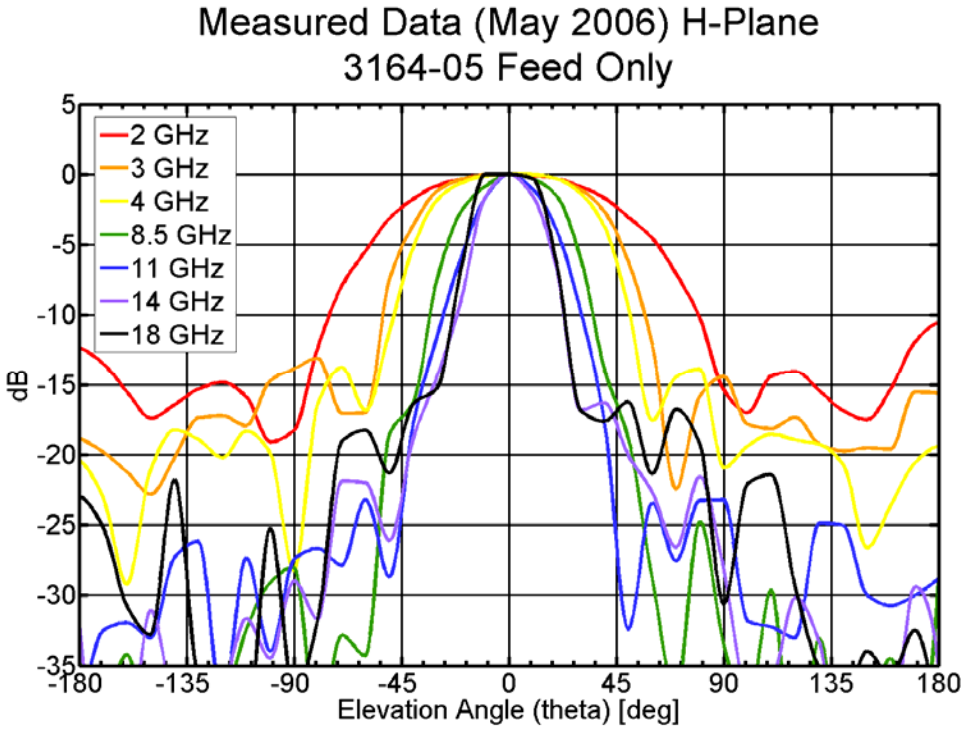
6.2.1 Far-Field Patterns for Feed Only

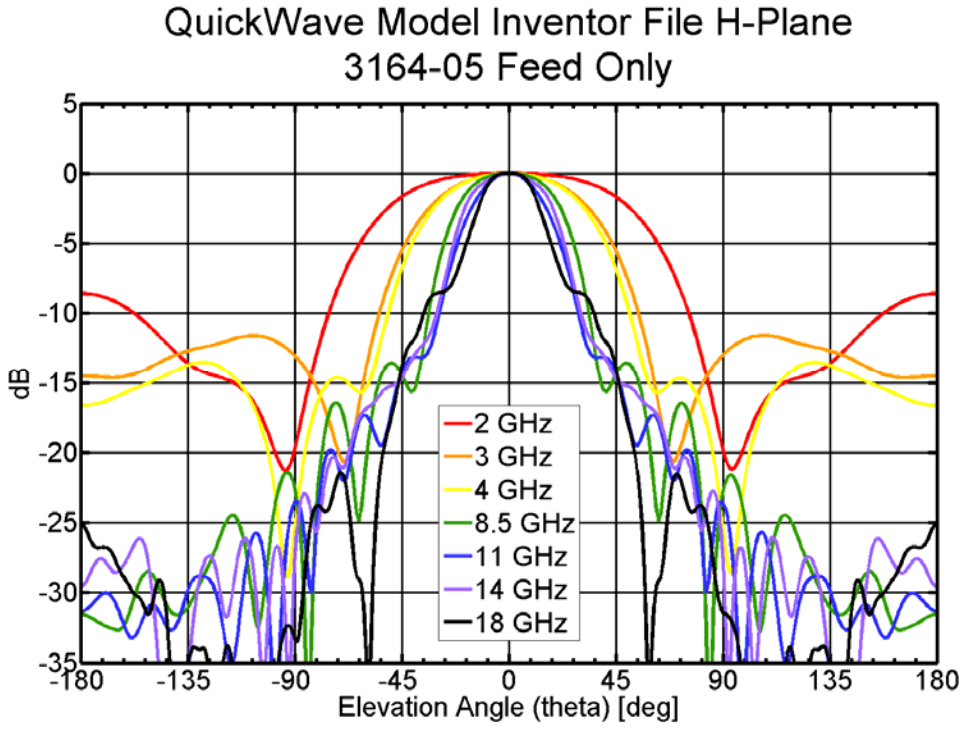
Comparison of Measured Data and QuickWave Model E-Plane Patterns:





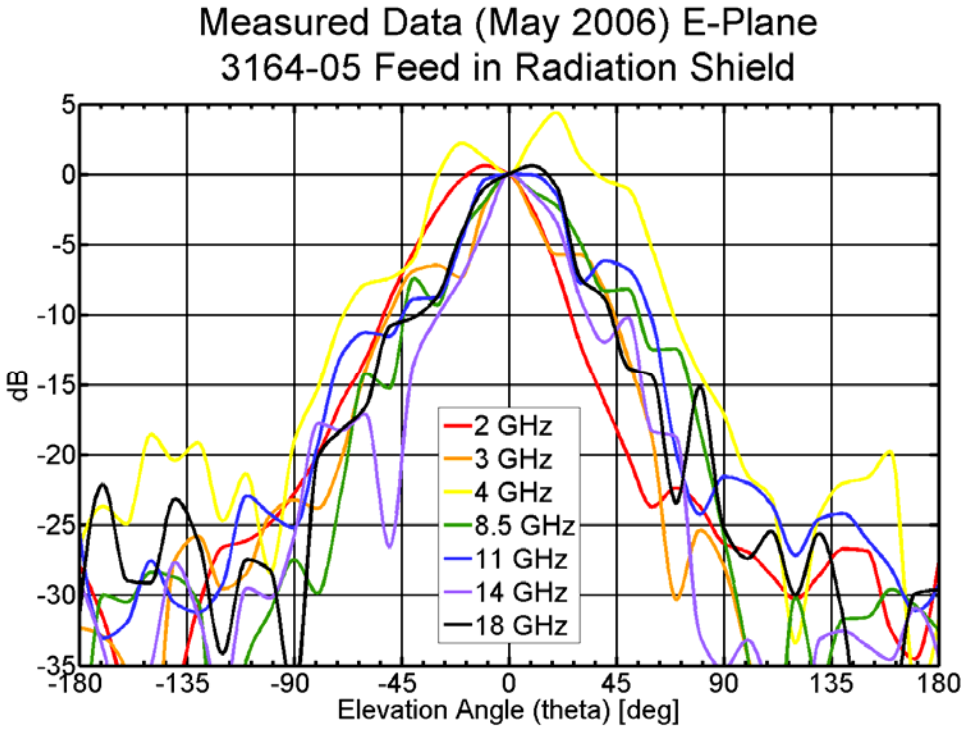
Comparison of Measured Data and QuickWave Model H-Plane Patterns:

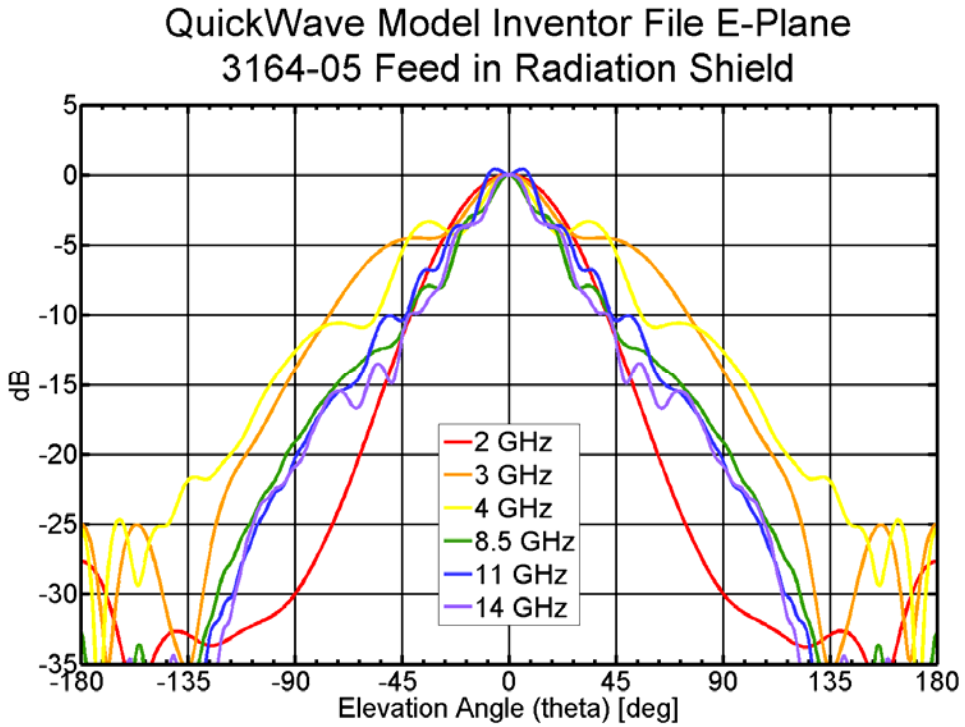




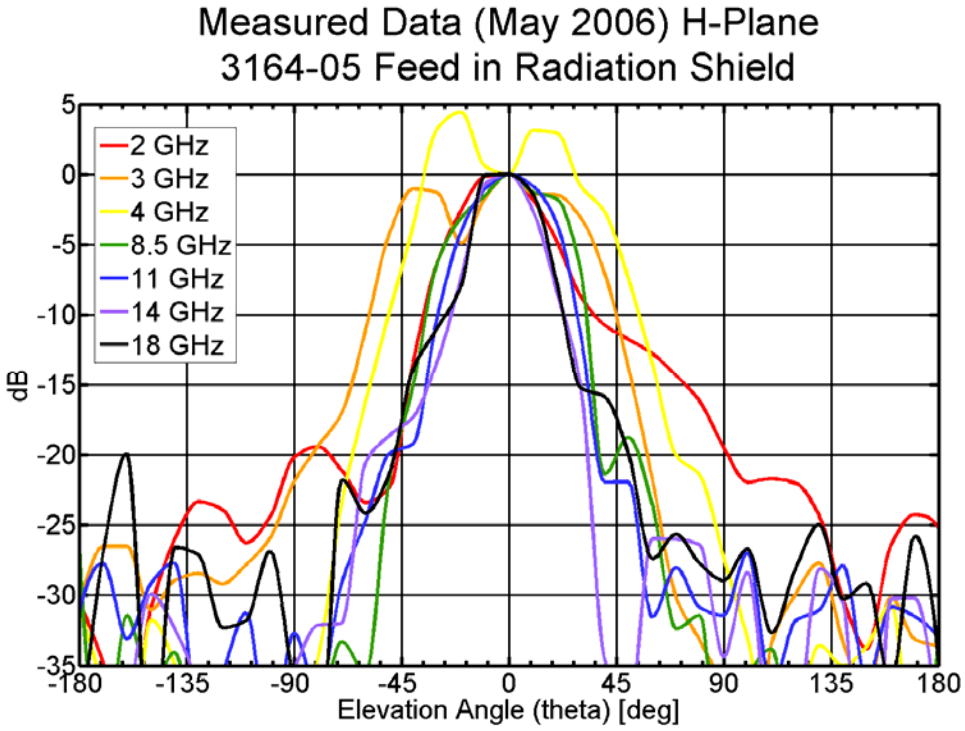
6.2.2 Far-Field Patterns for Feed in Radiation Shield

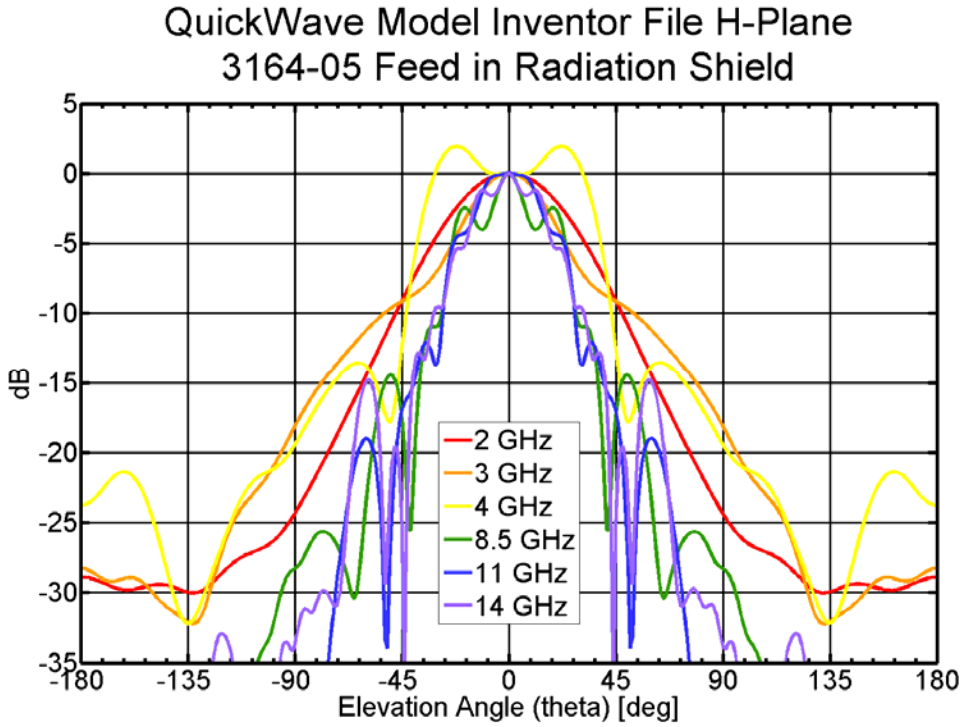
Comparison of Measured Data and QuickWave Model E-Plane Patterns:





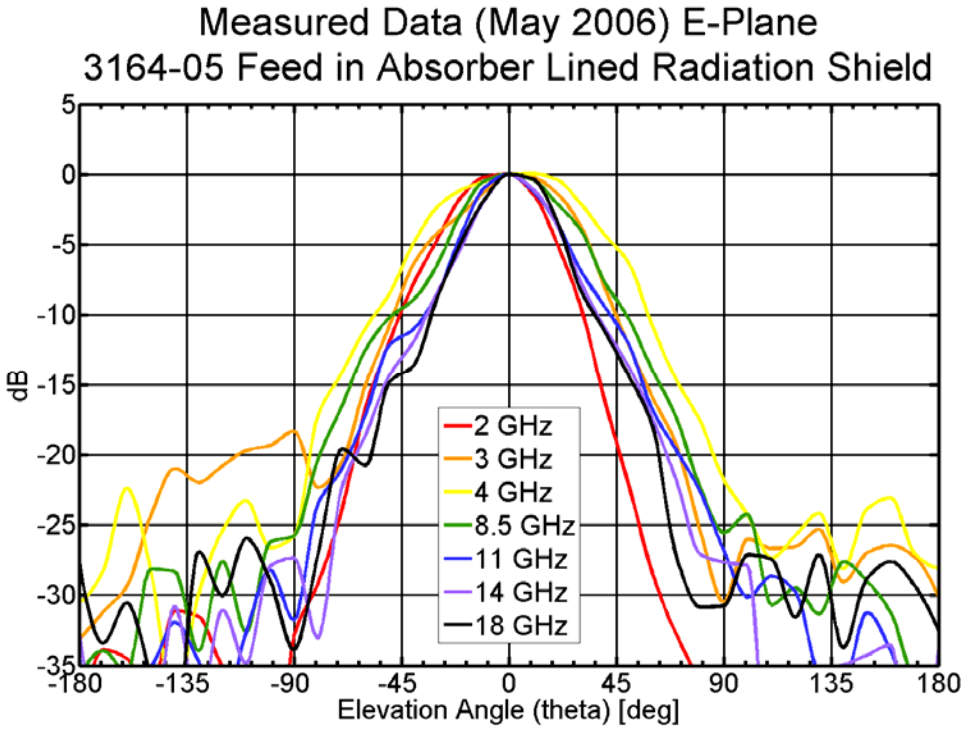
Comparison of Measured Data and QuickWave Model H-Plane Patterns:

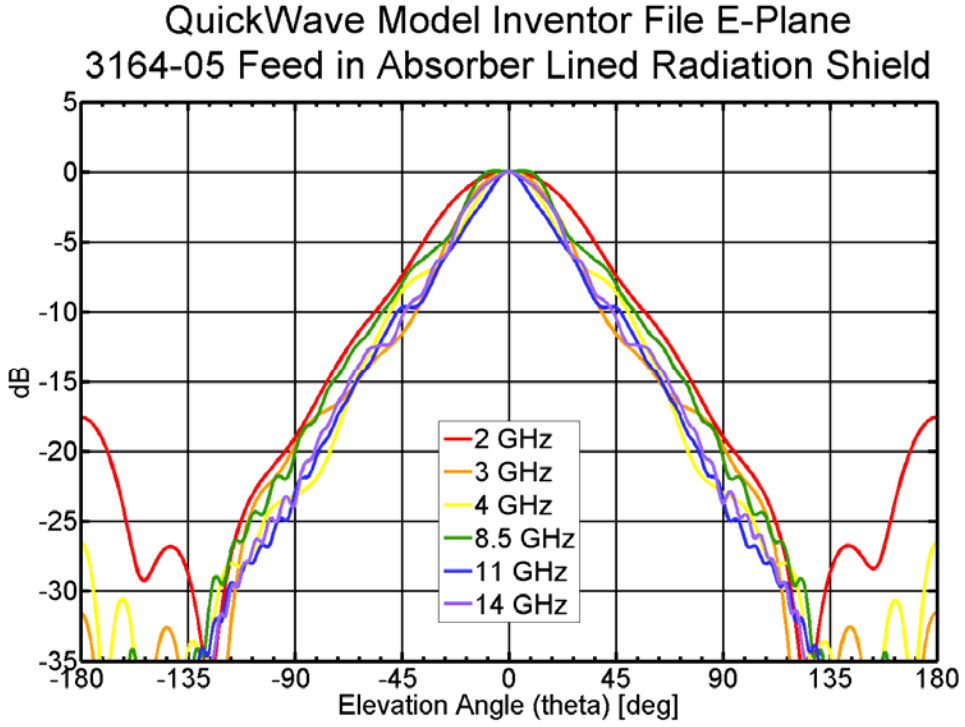




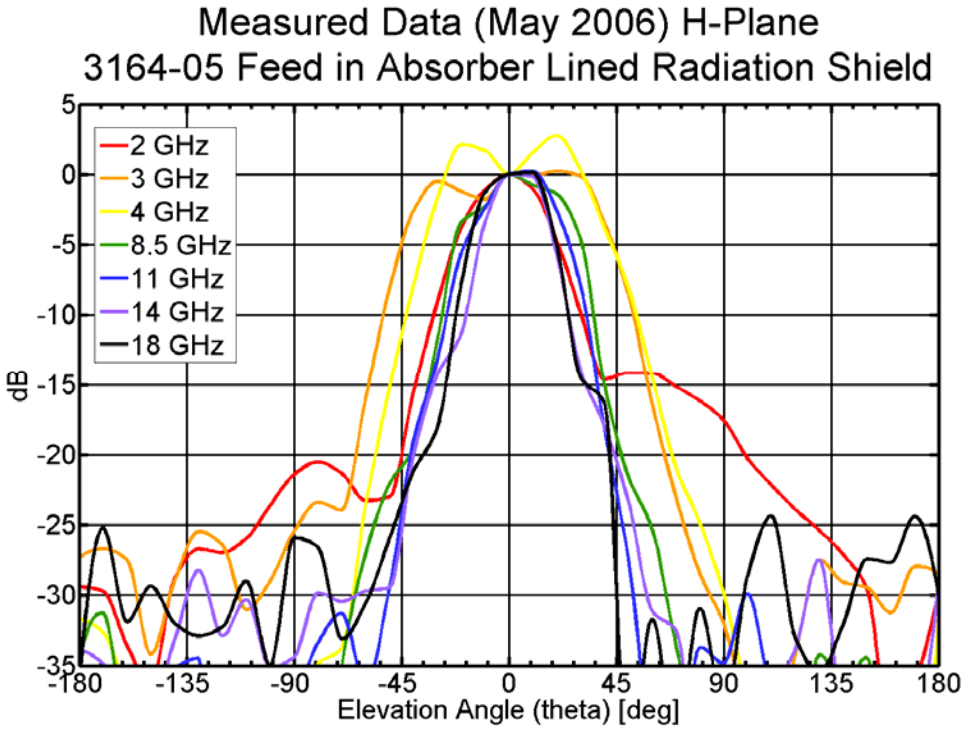
6.2.3 Far-Field Patterns for Feed in Absorber Lined Radiation Shield

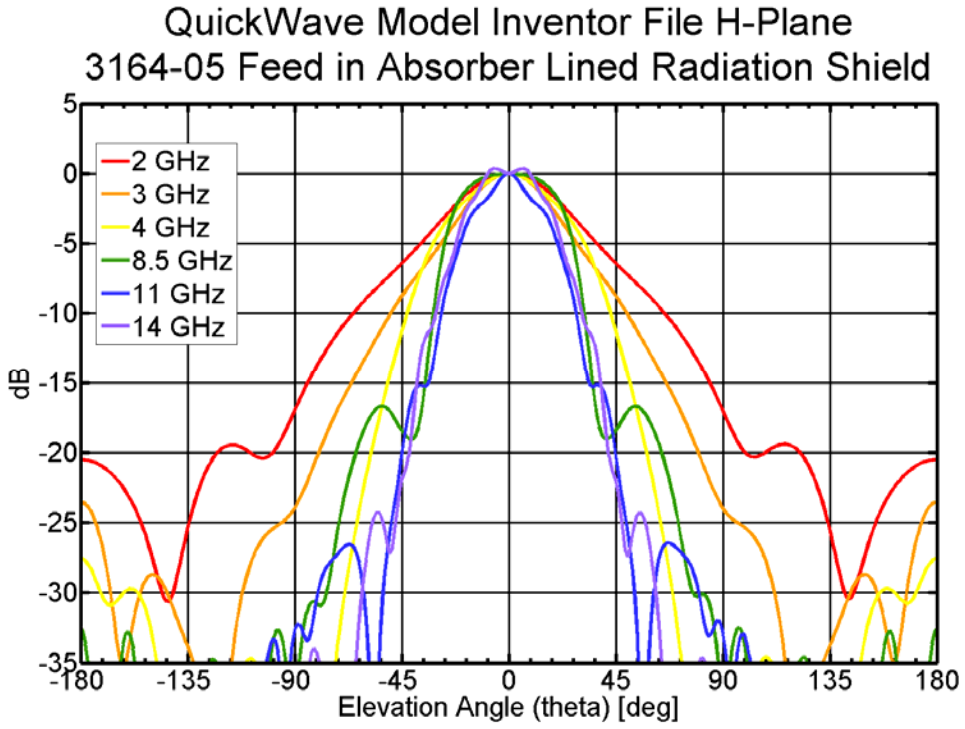
Comparison of Measured Data and QuickWave Model E-Plane Patterns:





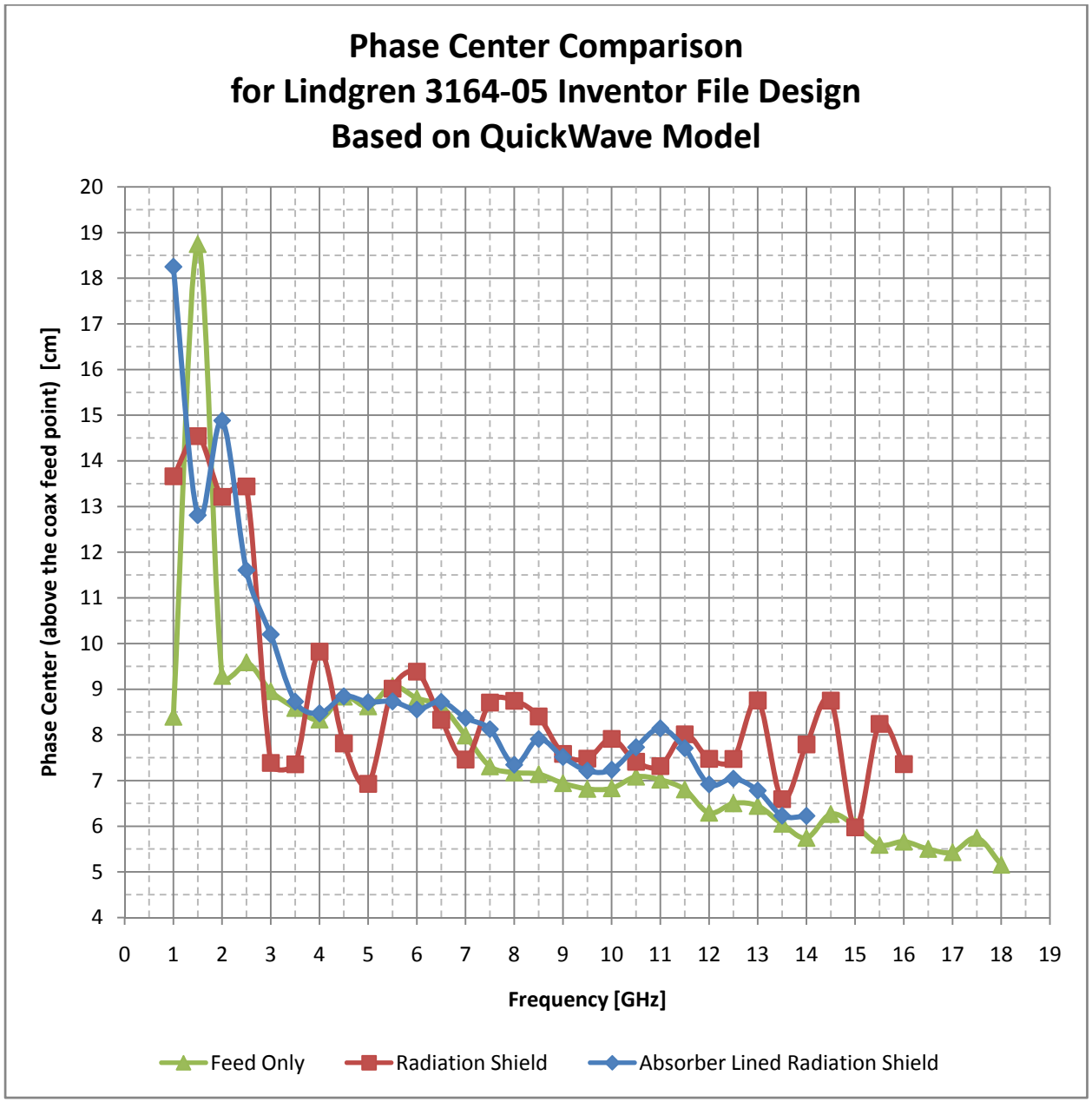
Comparison of Measured Data and QuickWave Model H-Plane Patterns:



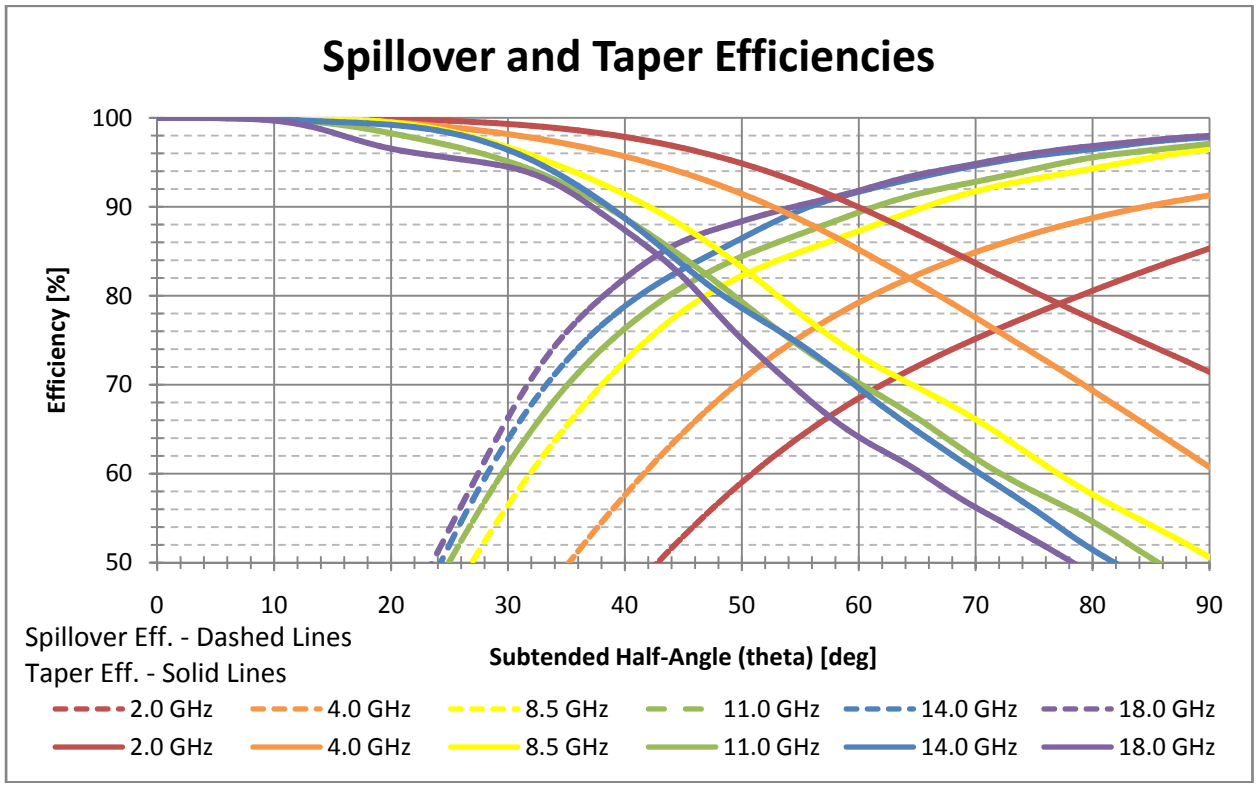


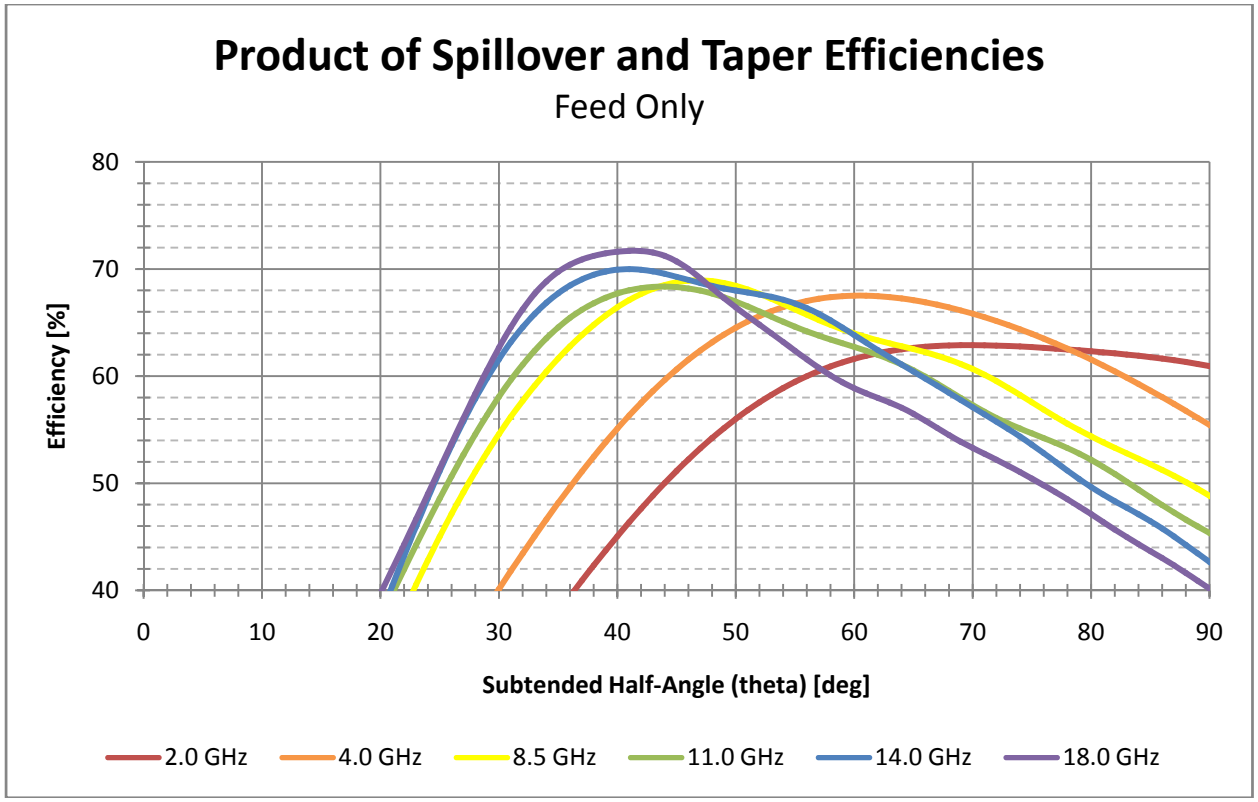
6.2.4 Phase Center

The phase center variation appears to be best using the absorber lined radiation shield.

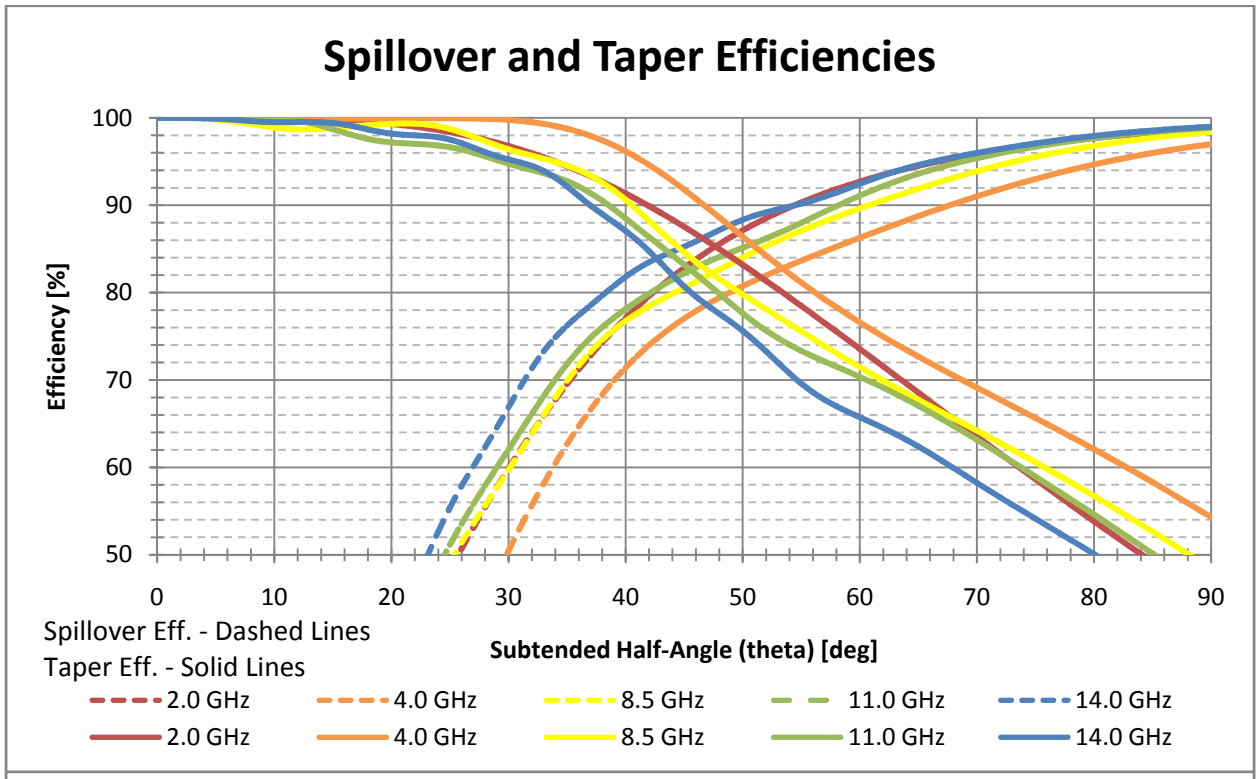


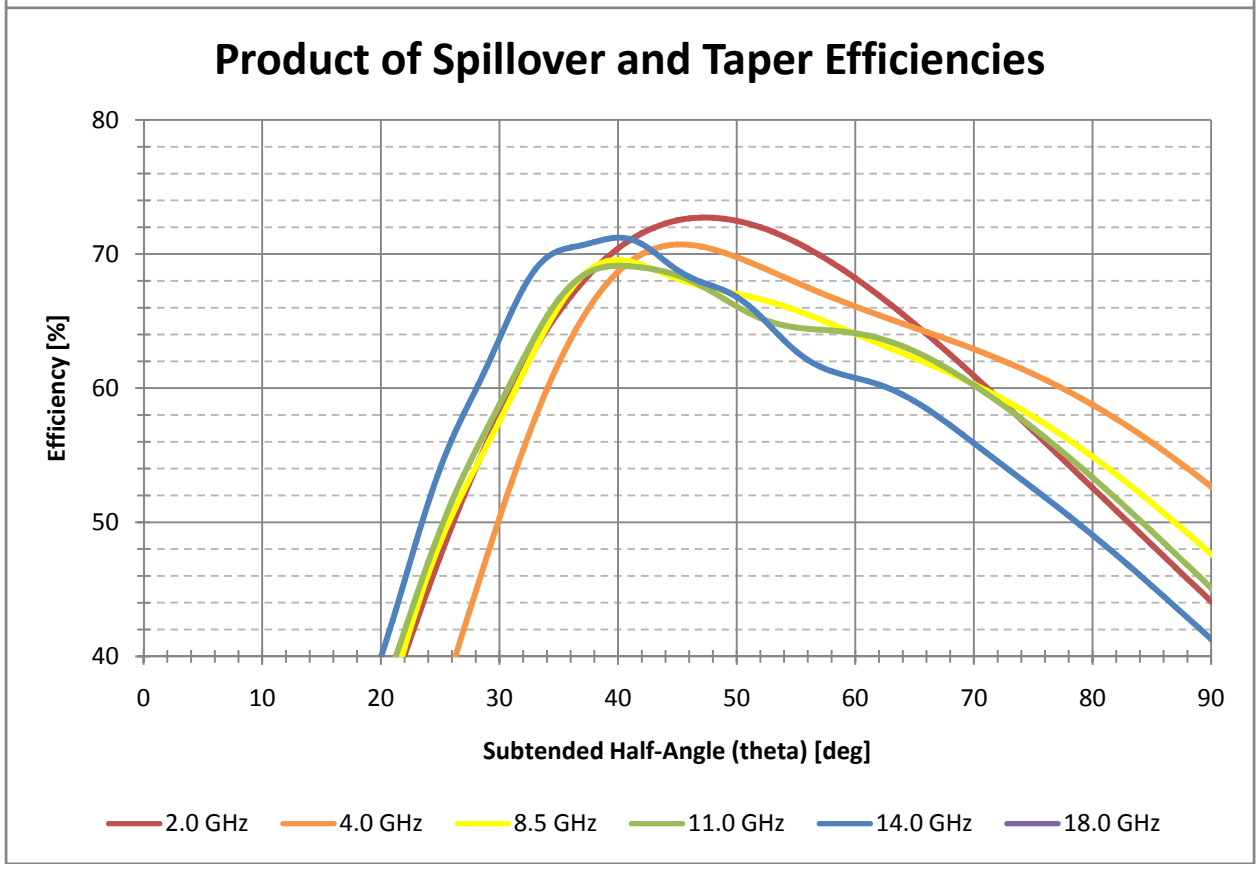
6.2.5 Efficiencies for Feed Only



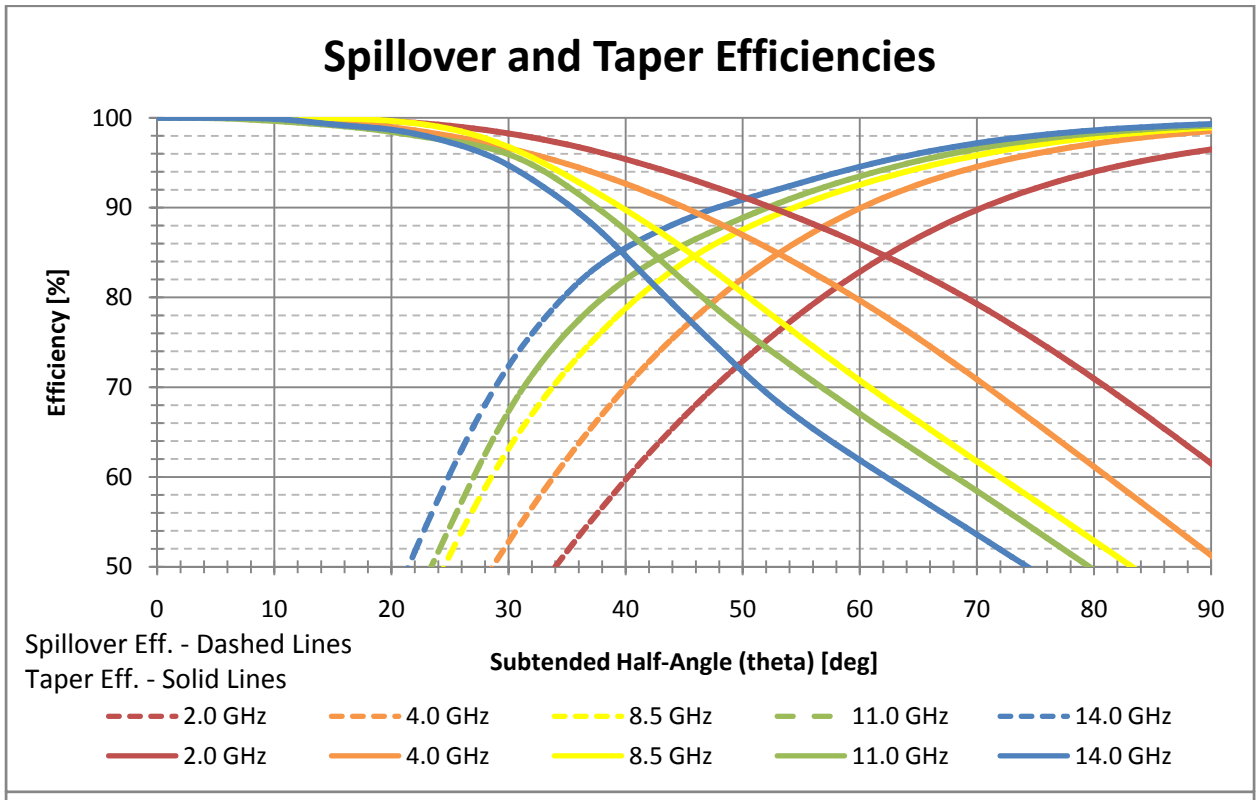


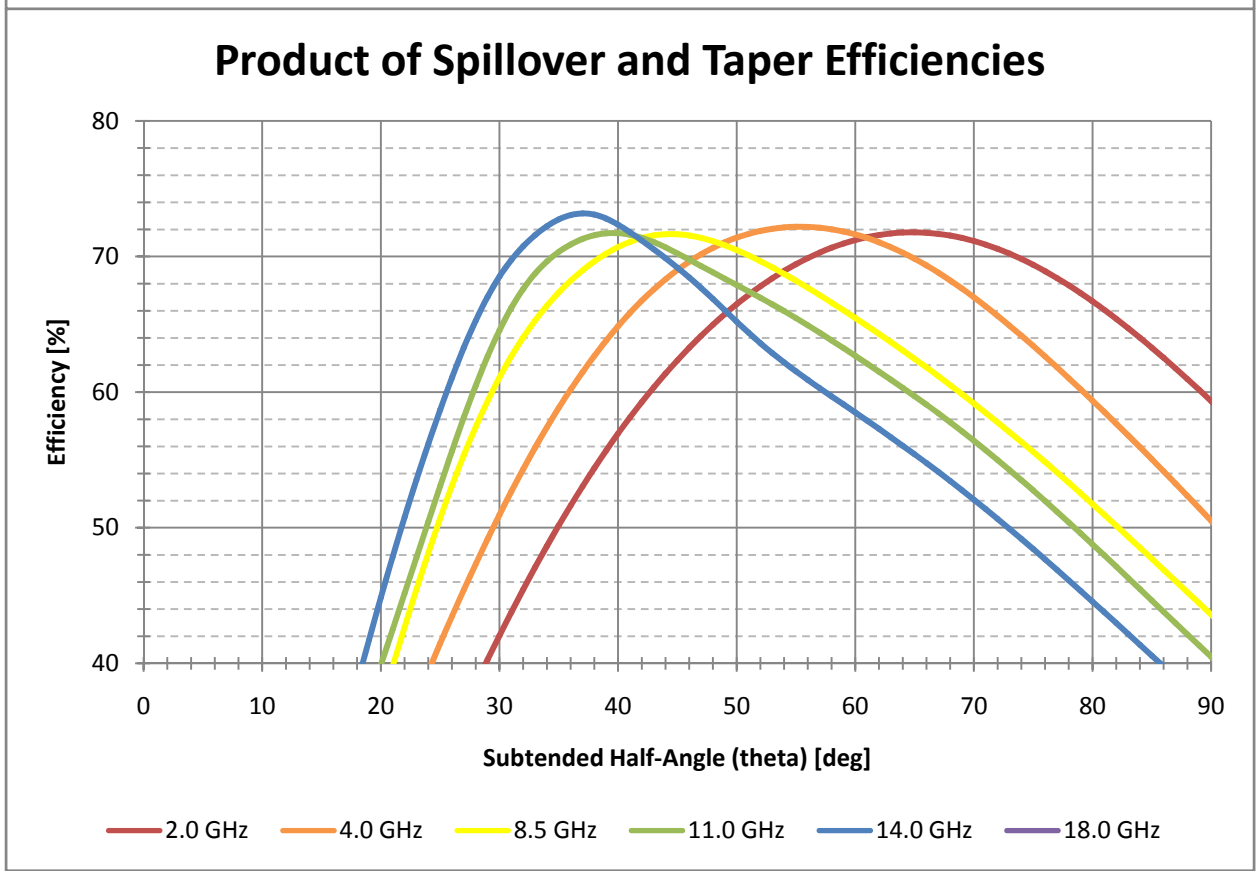
6.2.6 Efficiencies for Feed in Radiation Shield





6.2.7 Efficiencies for Feed in Absorber Lined Radiation Shield





6.3 Data from CST Model of 3164-05 Design (1-18 GHz)

The design of the ETS-Lindgren 3164-05 (see section 3.1.2) was modeled and simulated using CST MWS. The following data shows results from this simulation, in addition to a comparison to measured data from the actual Lindgren 3164-05 feed. The images below show the actual feed and the CST model side by side.

Lindgren 3164-05 Actual Feed

Figure 6.3 a

Lindgren 3164-05 CST Model

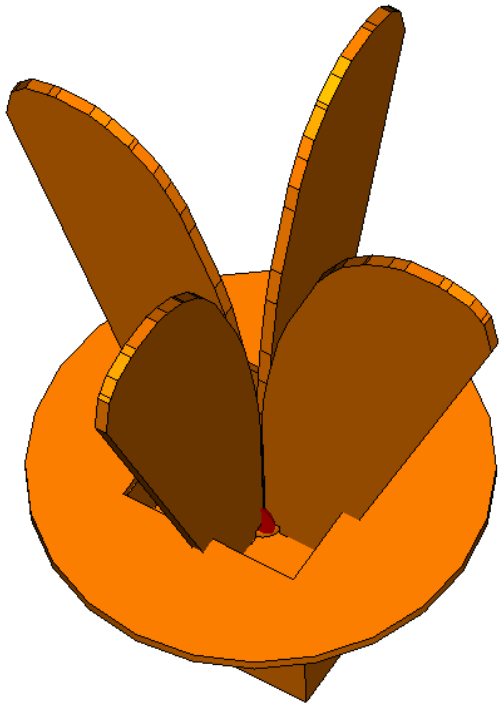
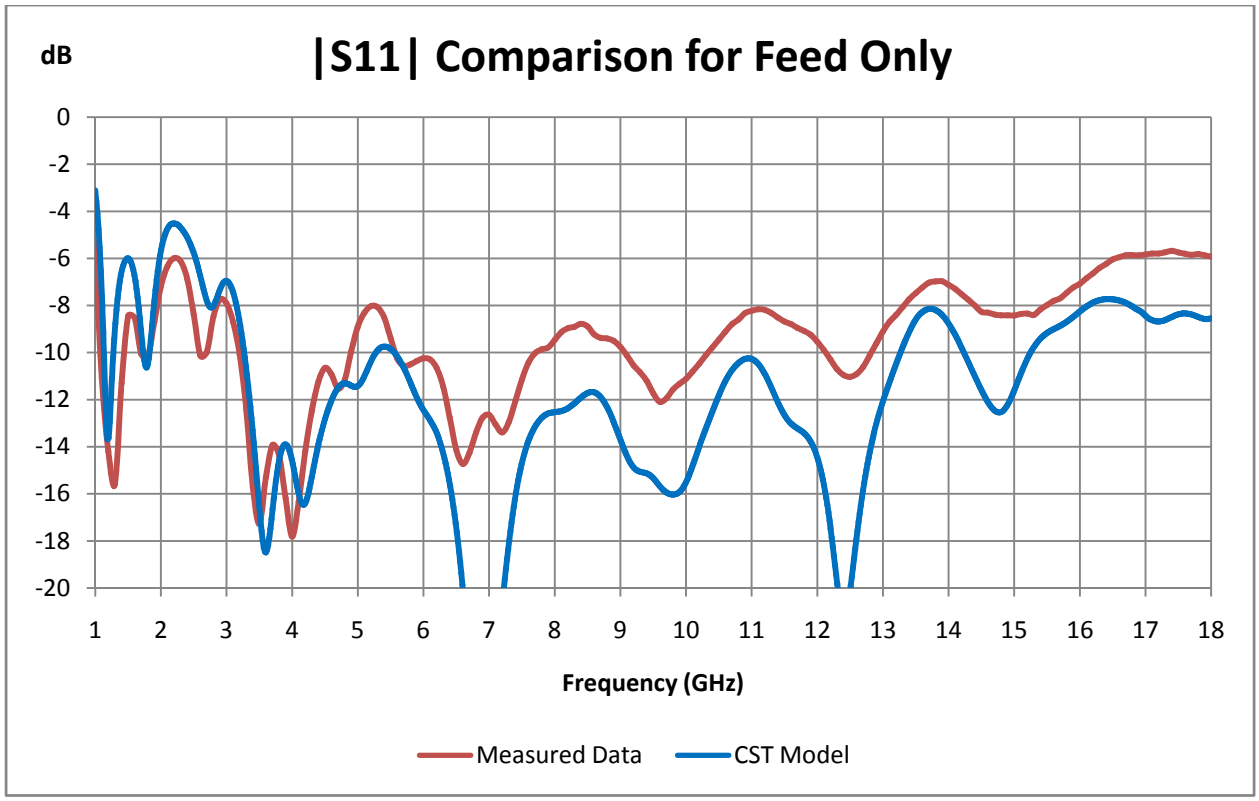


Figure 6.3 b

6.3.1 Return Loss

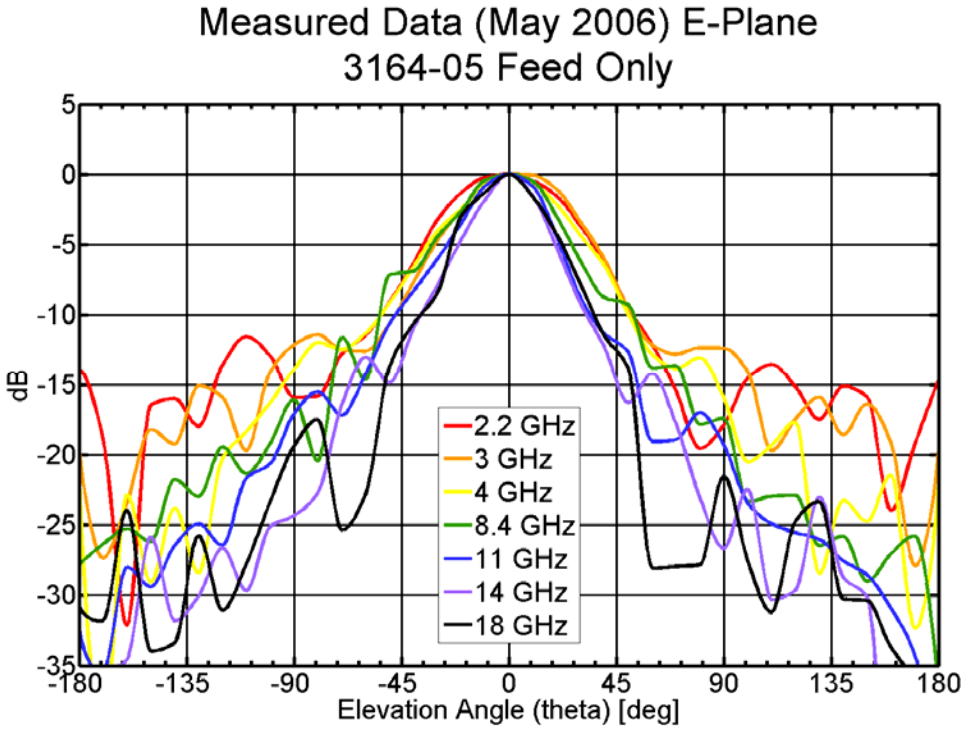
It is important to realize the differences between the model and the actual feed. The input of the actual feed (from which the measured data was taken) is a SMA connector that connects to coax going to a coax-to-slotline transition inside the slot between the fins. The CST model does not have this complicated input network. The model is fed by a simple E-field excitation across the slot at the same point that the coax excitation would be located. This should explain some differences between the modeled return loss and the measured data, because the model will not reflect losses in the coax or coax-to-slotline transition.

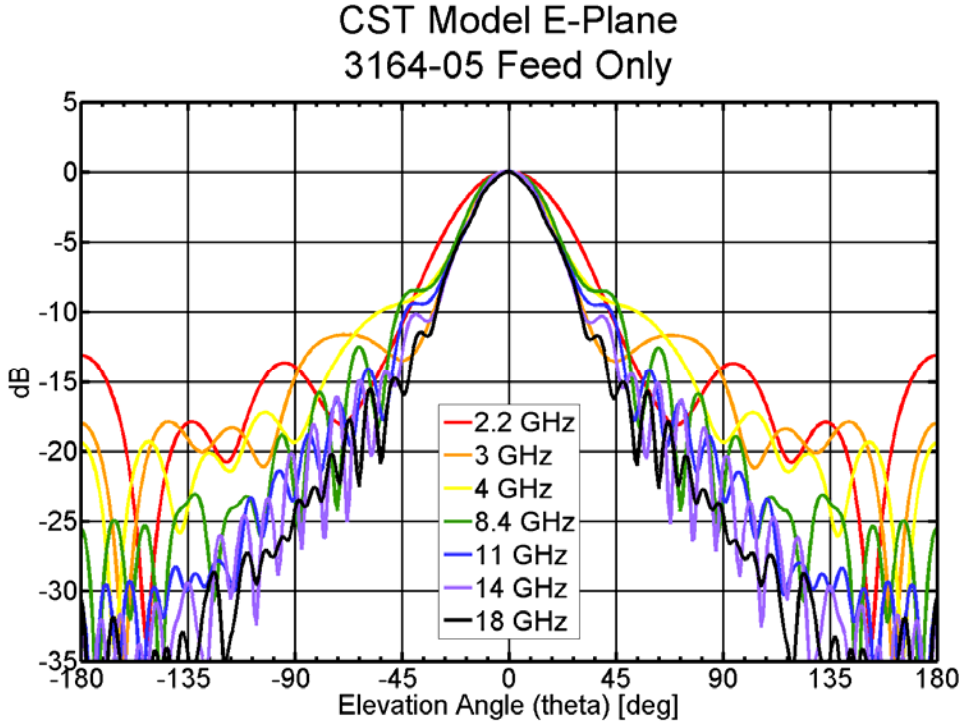


The model shows a surprising similarity good match to the measured values.

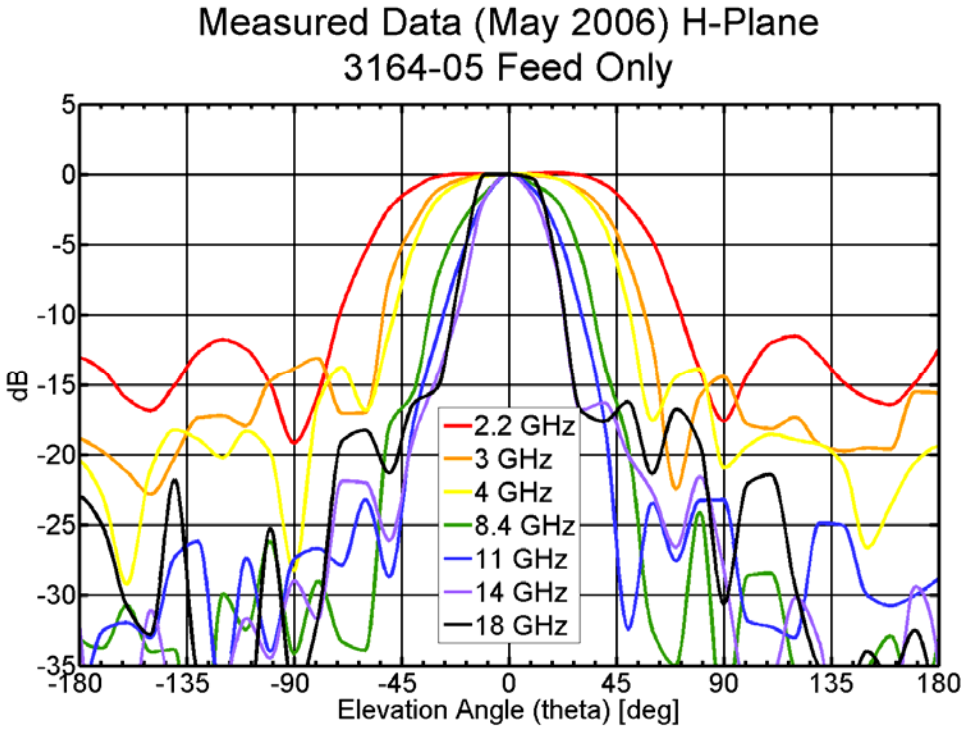
6.3.2 Far-Field Patterns for Feed Only

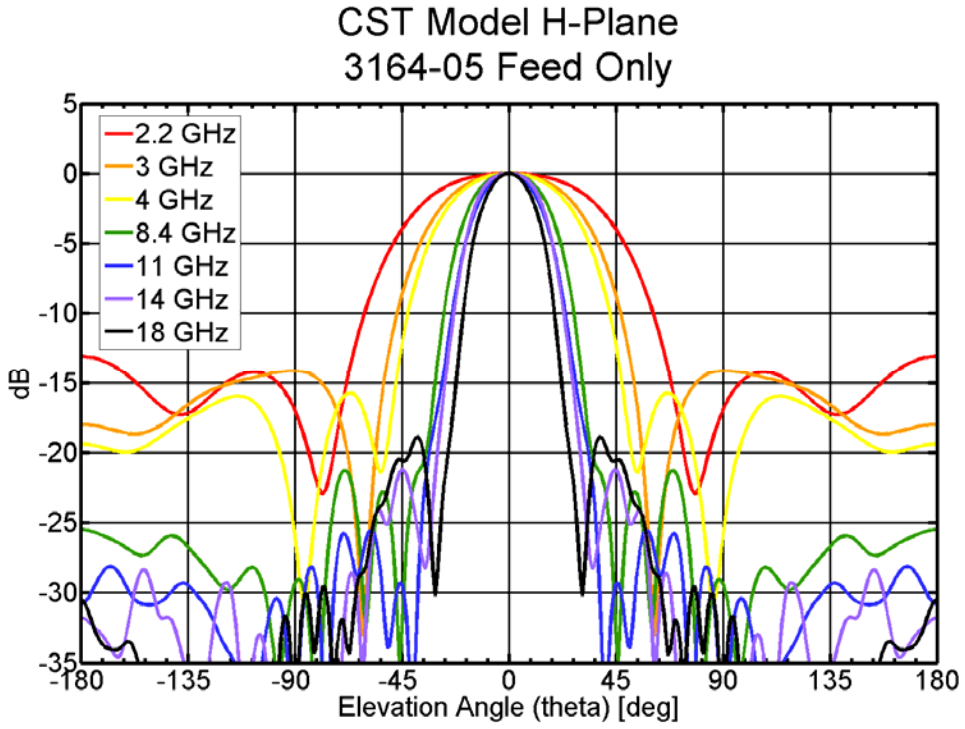
Comparison of Measured Data and CST Model E-Plane Patterns:



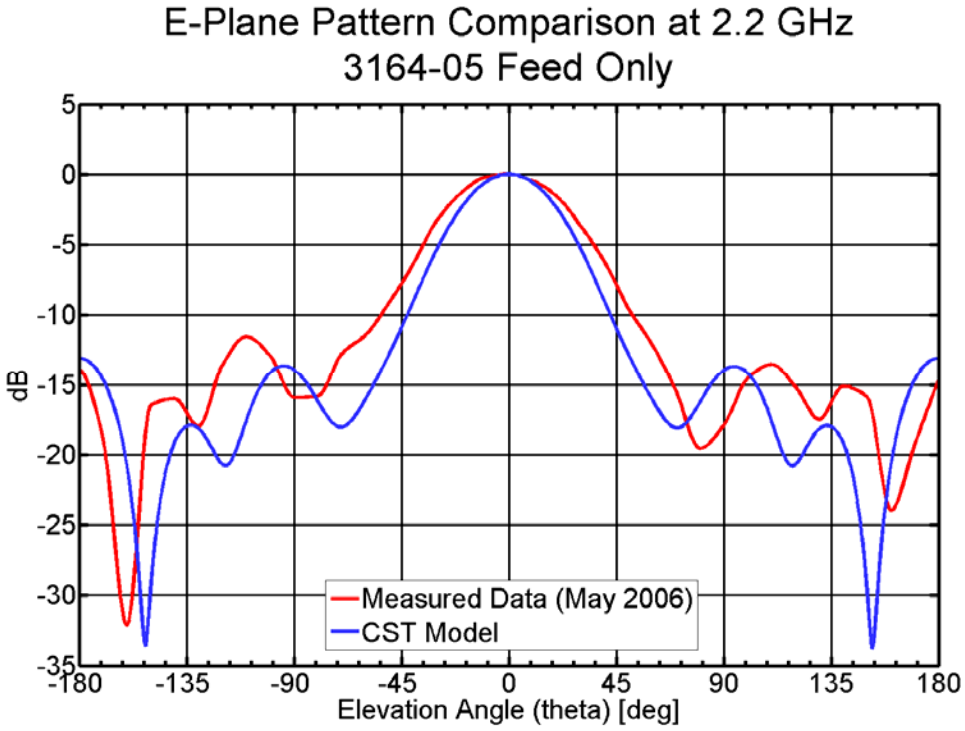


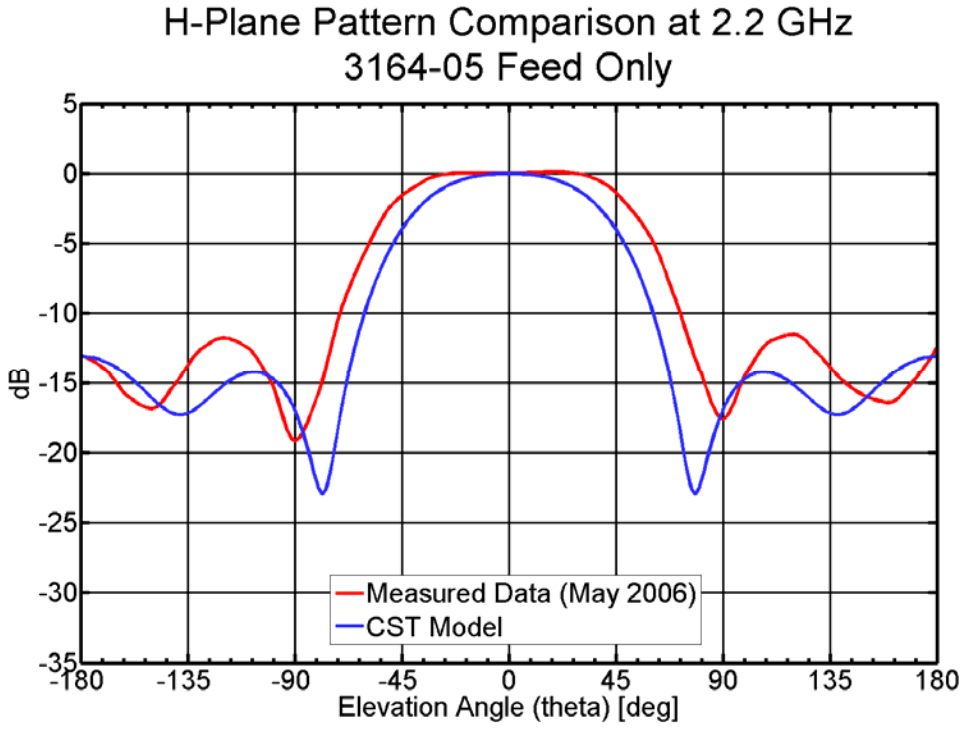
Comparison of Measured Data and CST Model H-Plane Patterns:



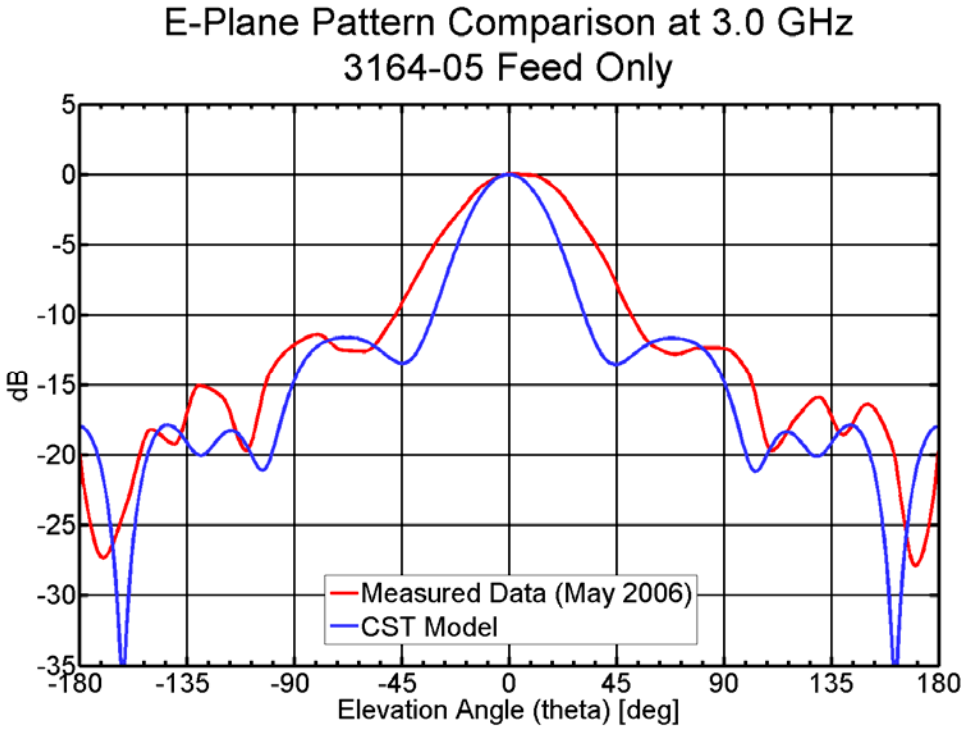


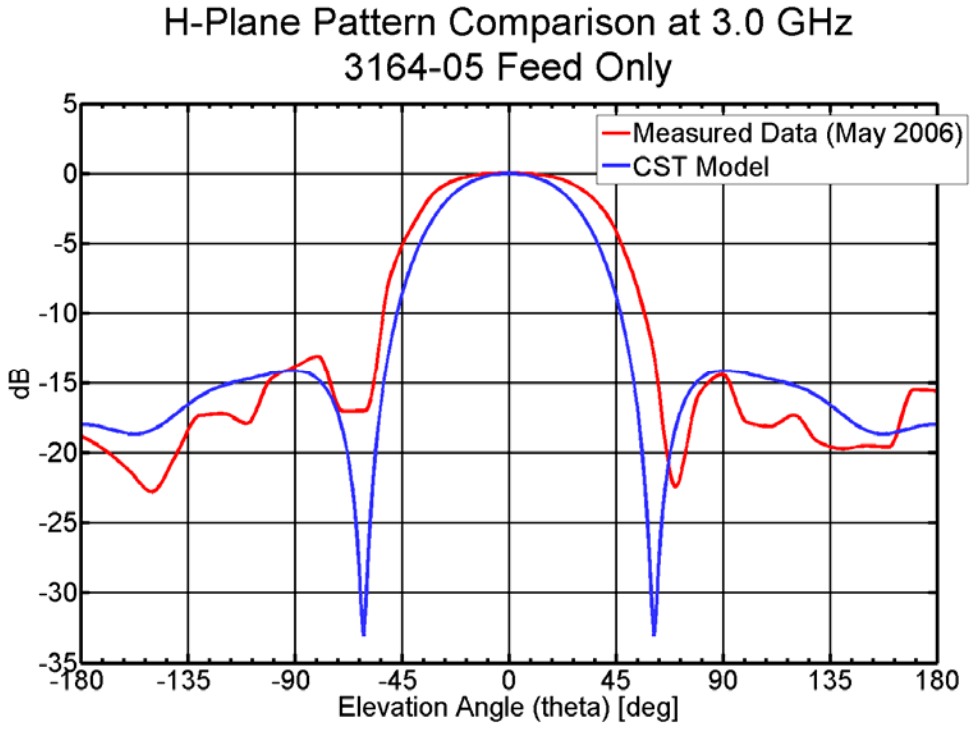
Comparison of Measured Data and CST Model Patterns at 2.2 GHz:



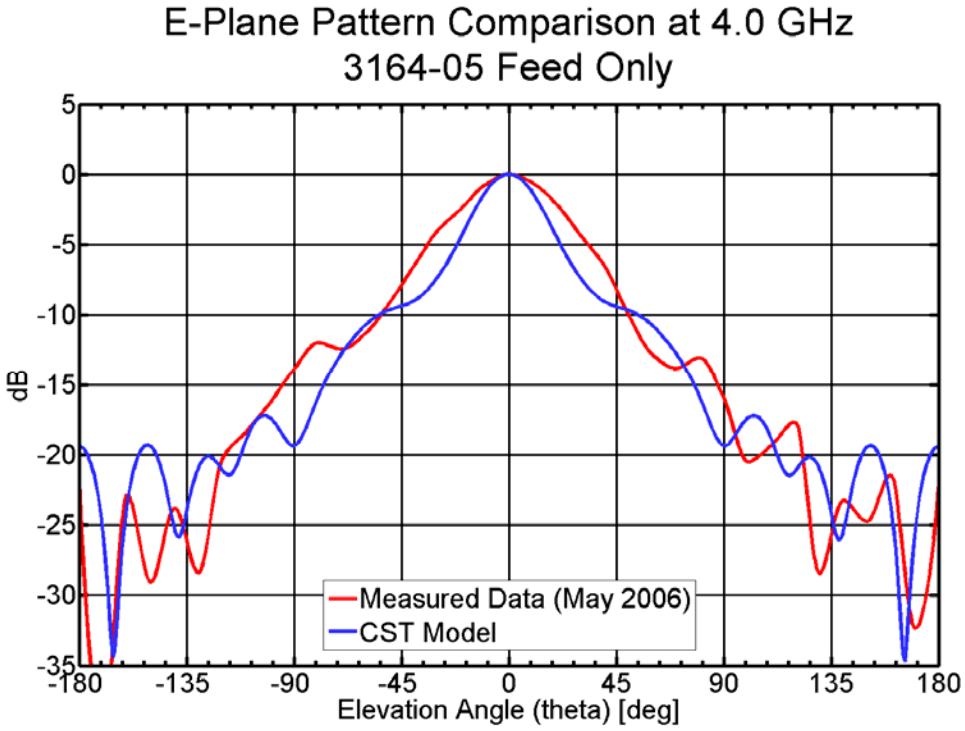


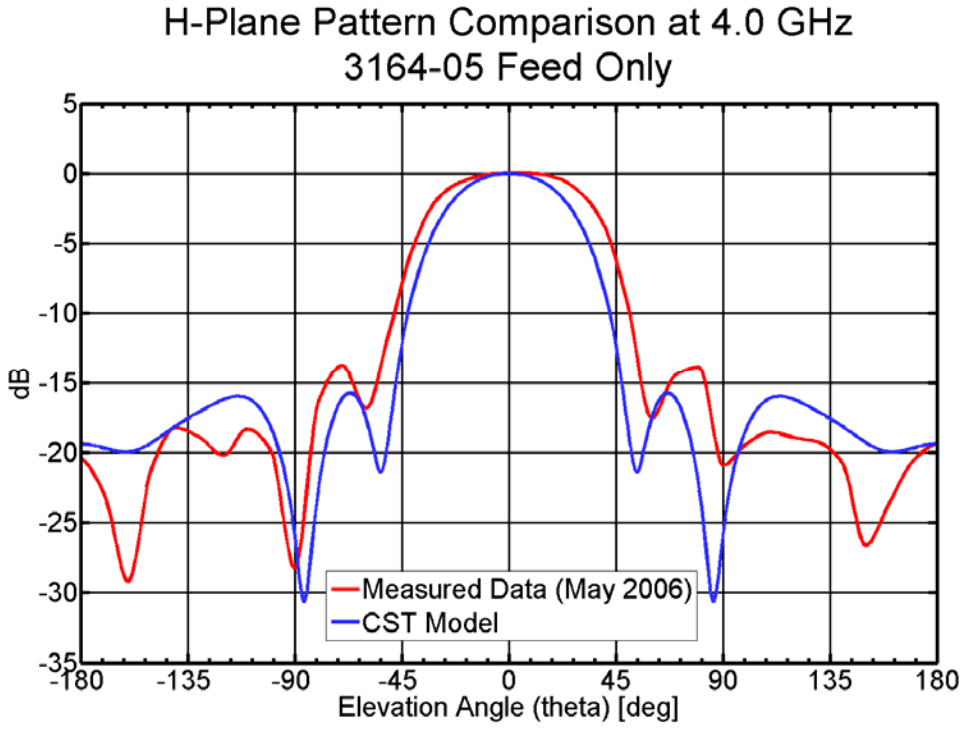
Comparison of Measured Data and CST Model Patterns at 3.0 GHz:



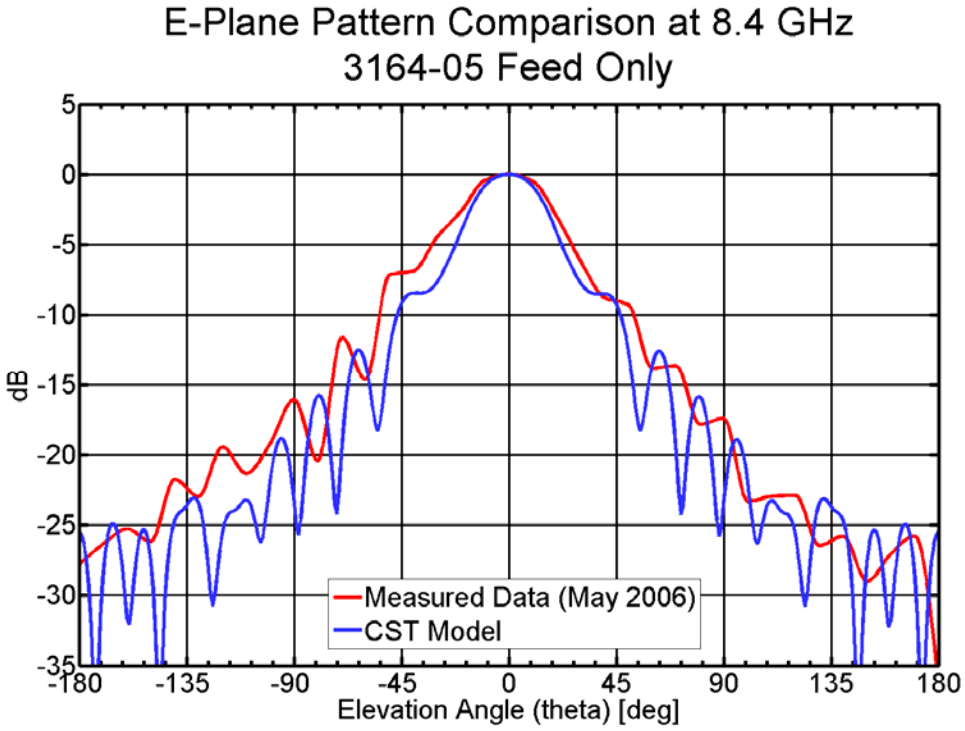


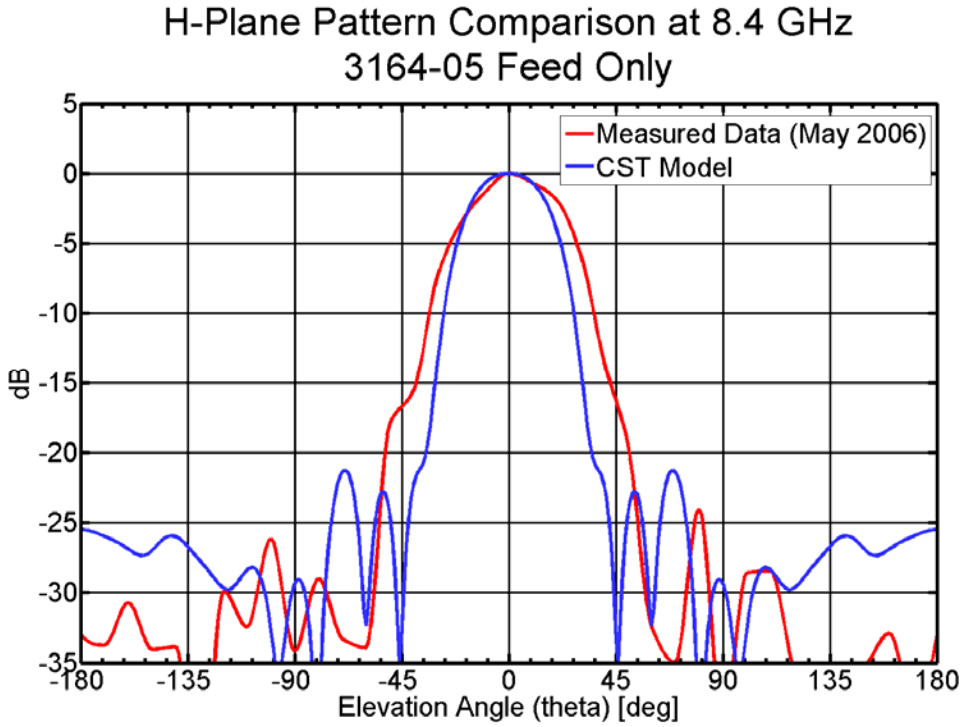
Comparison of Measured Data and CST Model Patterns at 4.0 GHz:



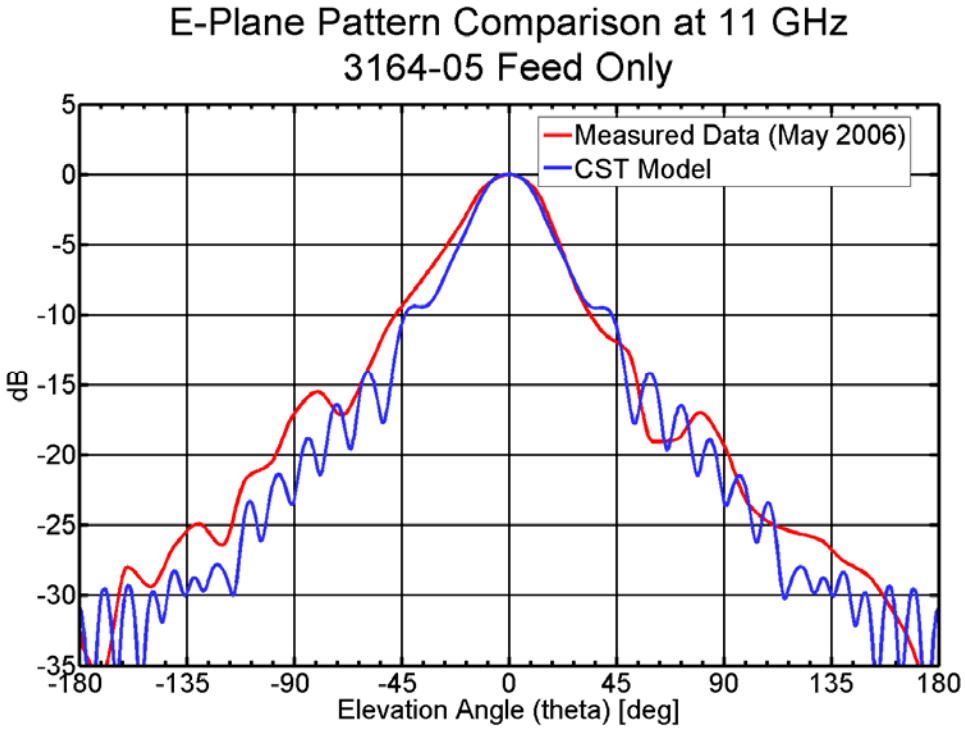


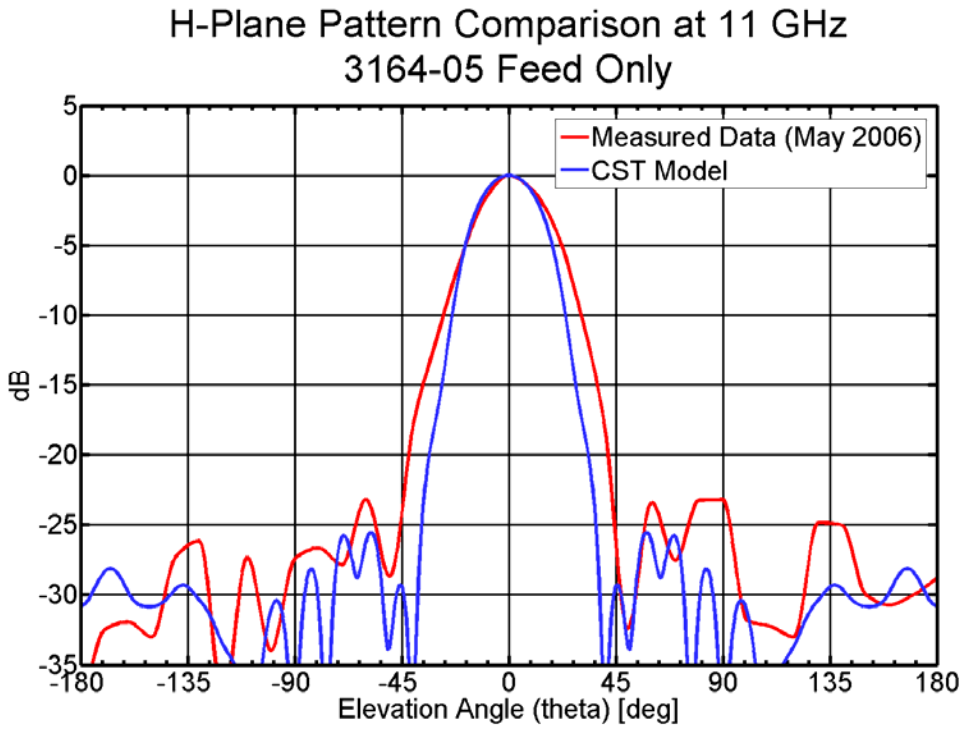
Comparison of Measured Data and CST Model Patterns at 8.4 GHz:



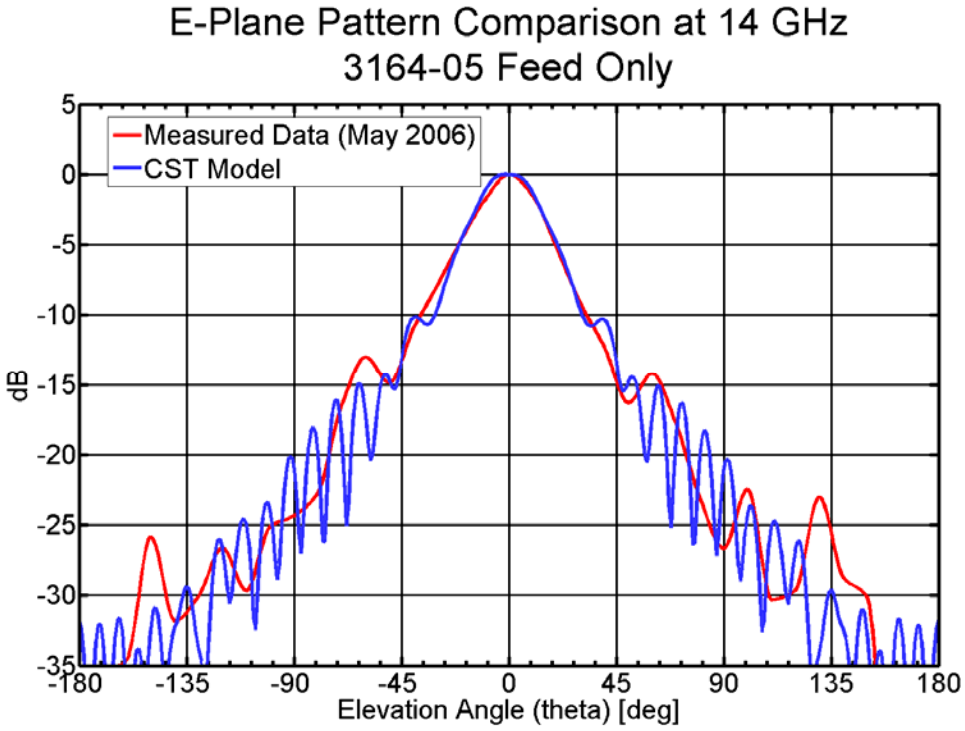


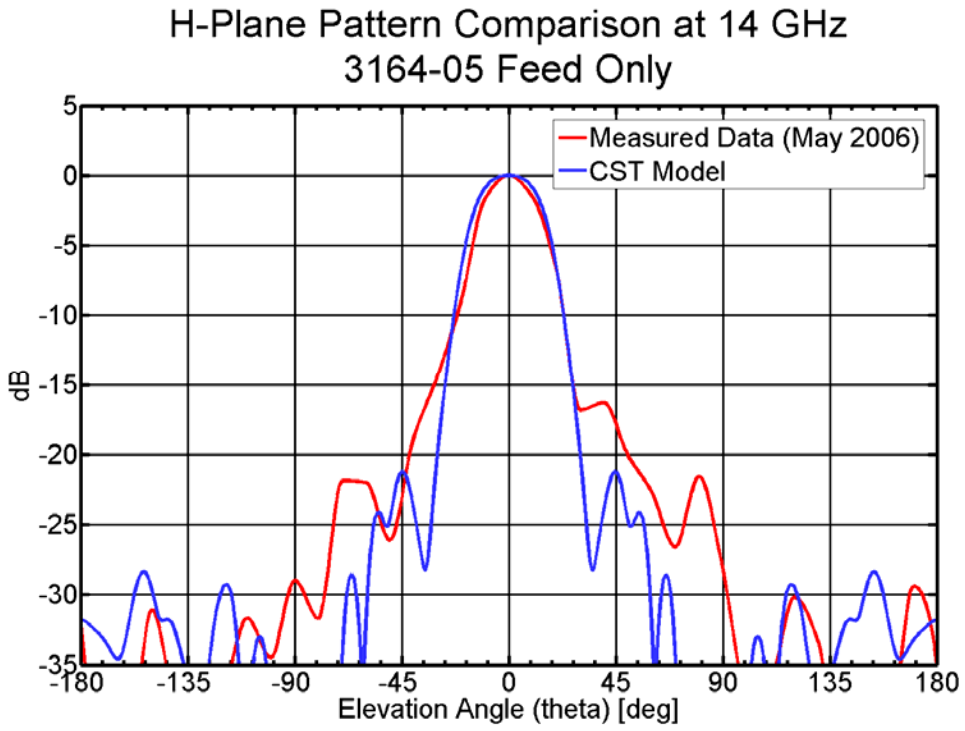
Comparison of Measured Data and CST Model Patterns at 11 GHz:



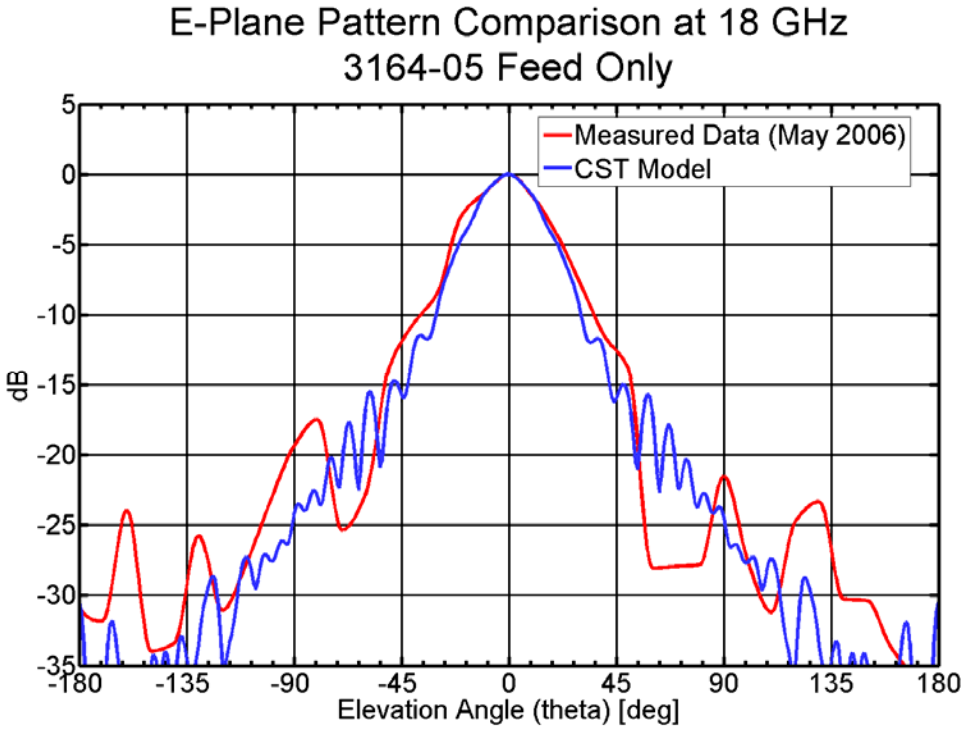


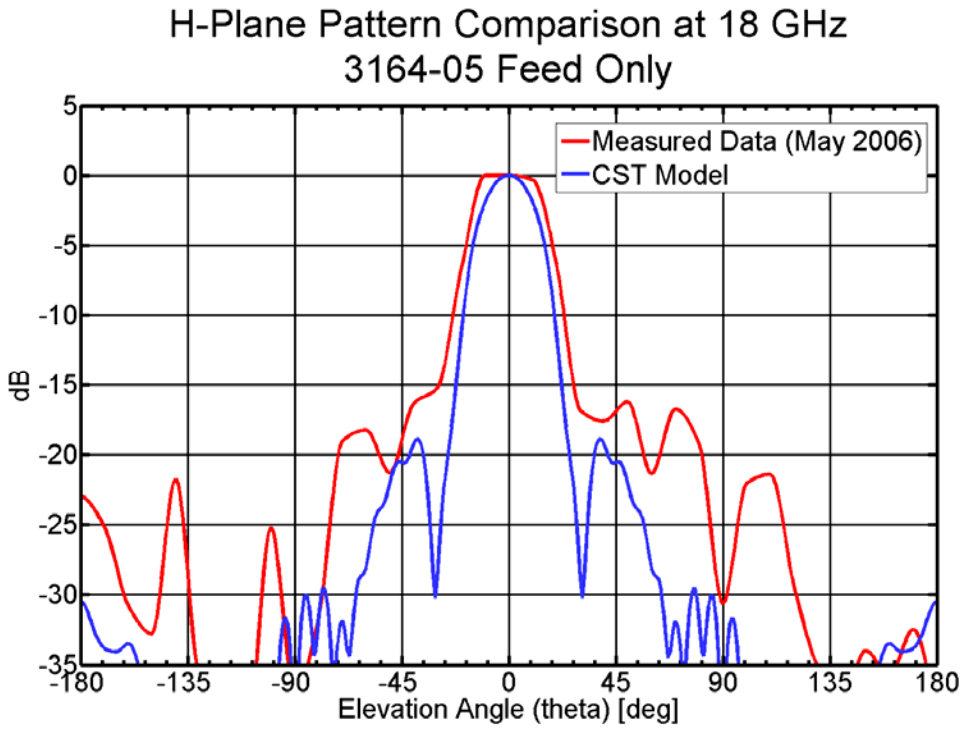
Comparison of Measured Data and CST Model Patterns at 14 GHz:





Comparison of Measured Data and CST Model Patterns at 18 GHz:





6.3.3 Far-Field Patterns for Feed in Radiation Shield

The images below show the actual feed in the radiation shield and the CST model.

Lindgren 3164-05 Actual Feed

Figure 6.3.3 a

Lindgren 3164-05 CST Model

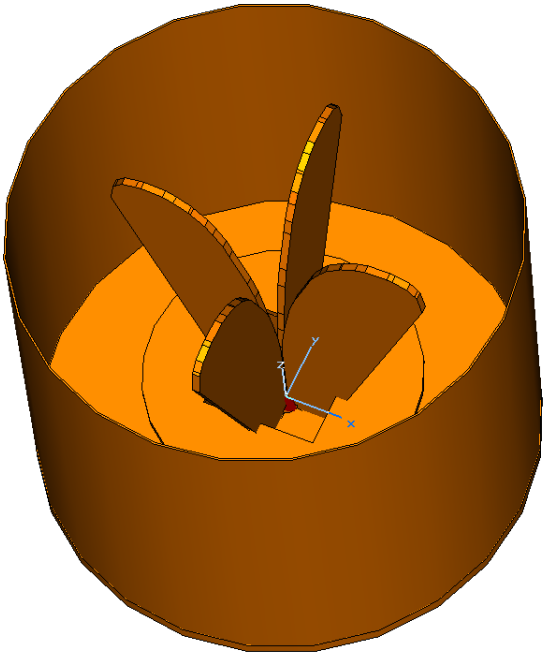
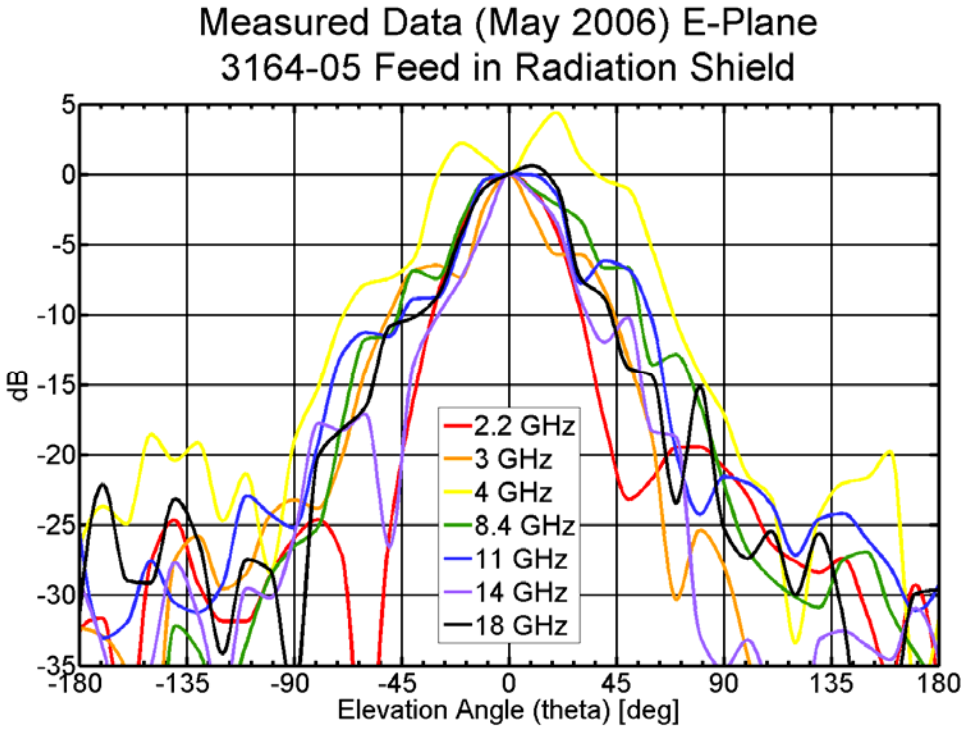
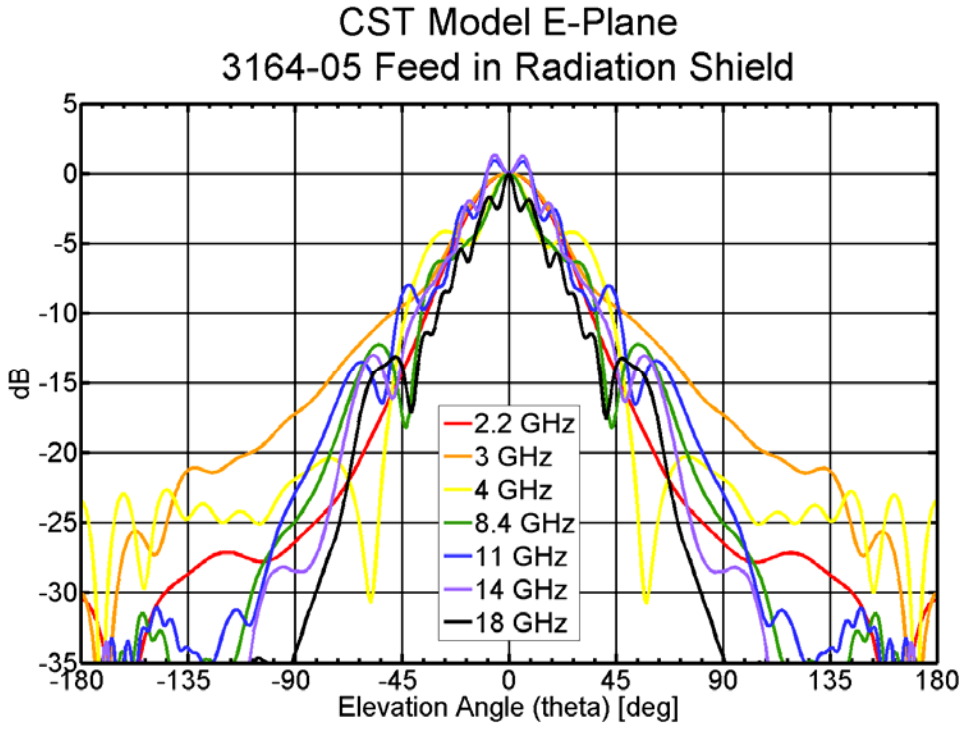


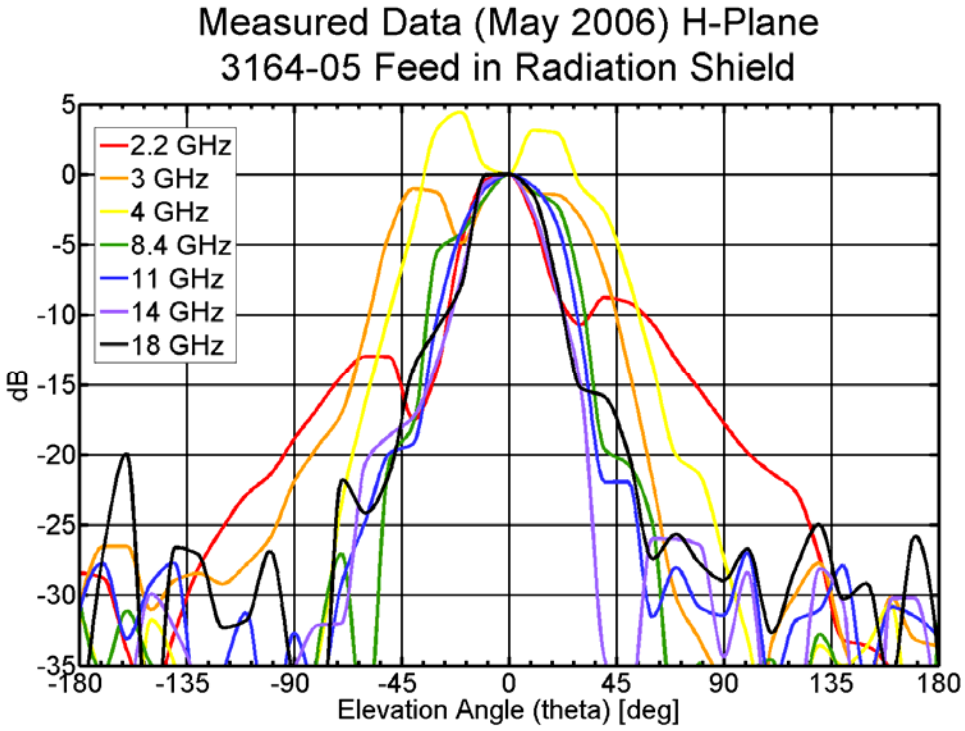
Figure 6.3.3 b

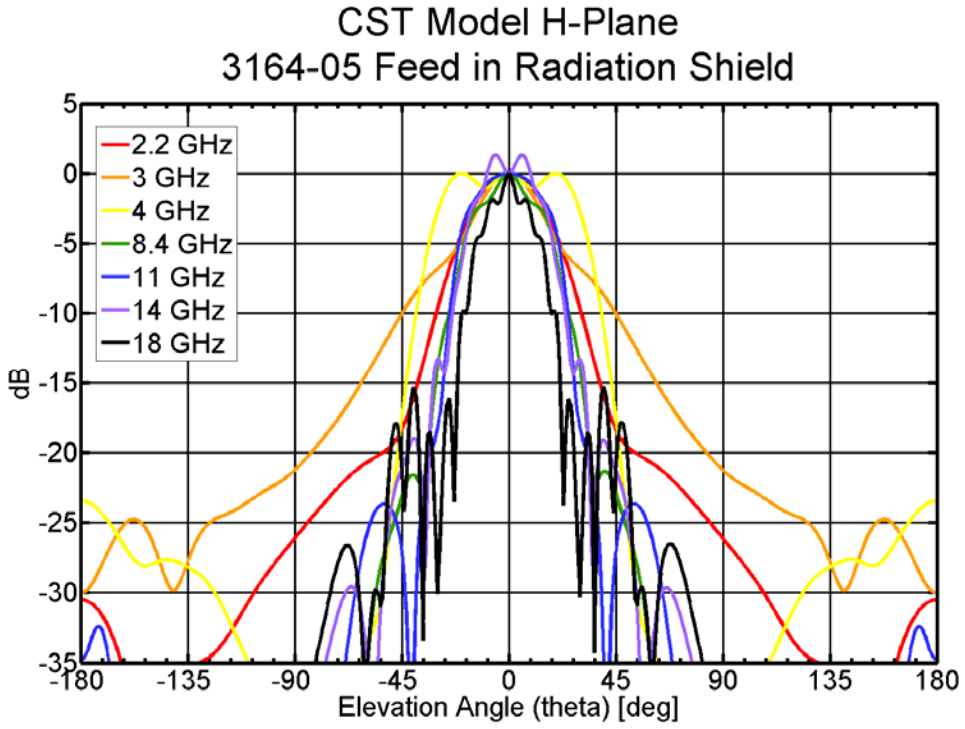
Comparison of Measured Data and CST Model E-Plane Patterns:



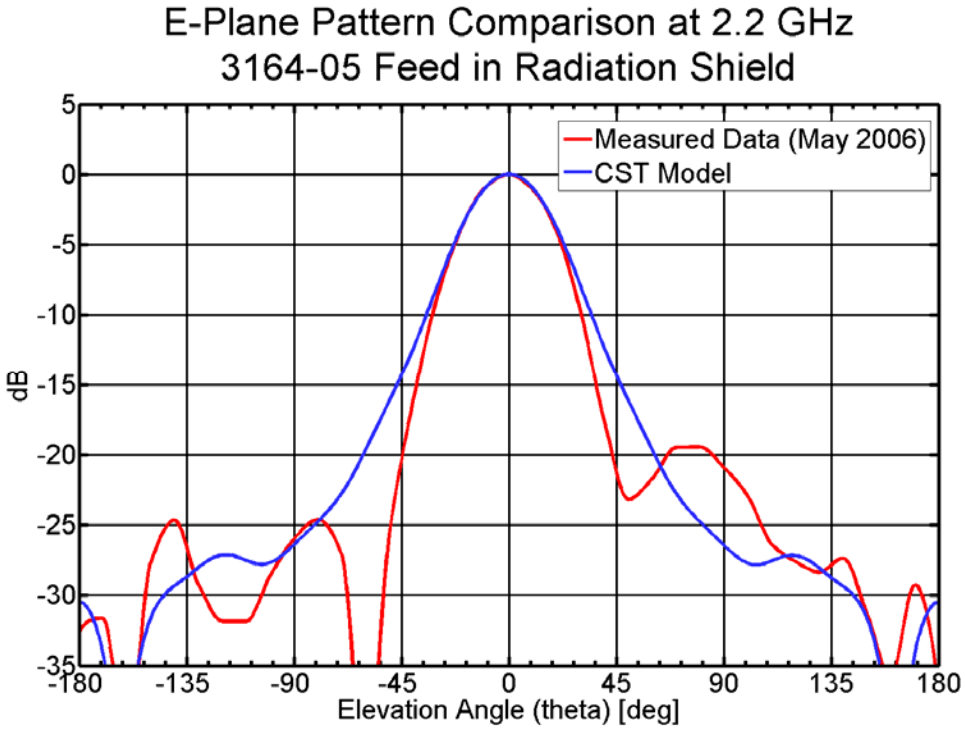


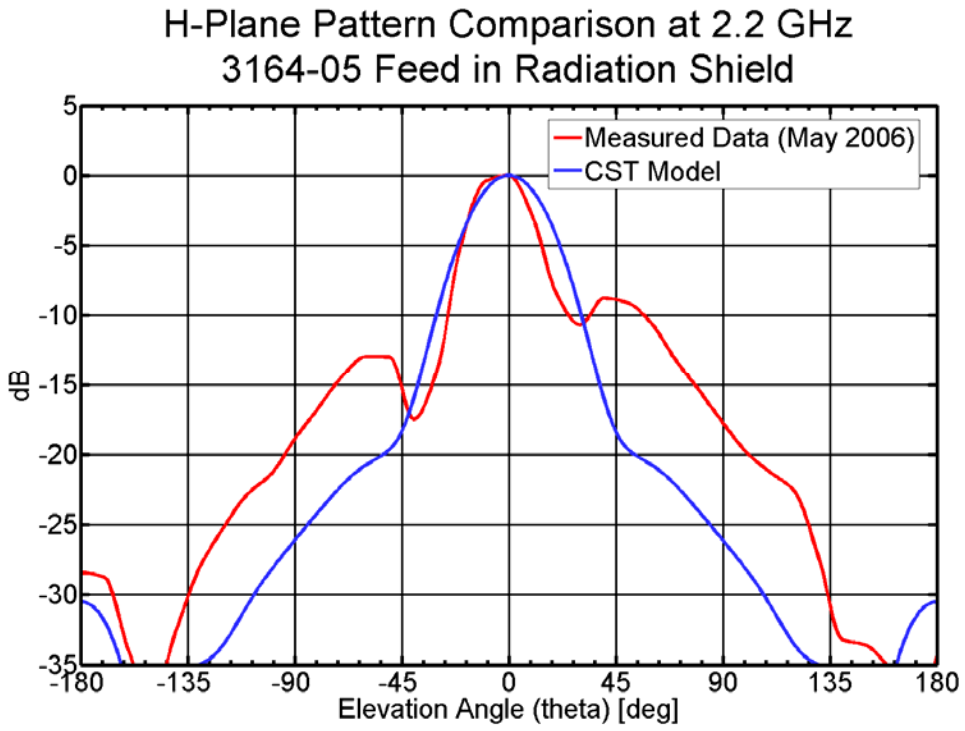
Comparison of Measured Data and CST Model H-Plane Patterns:



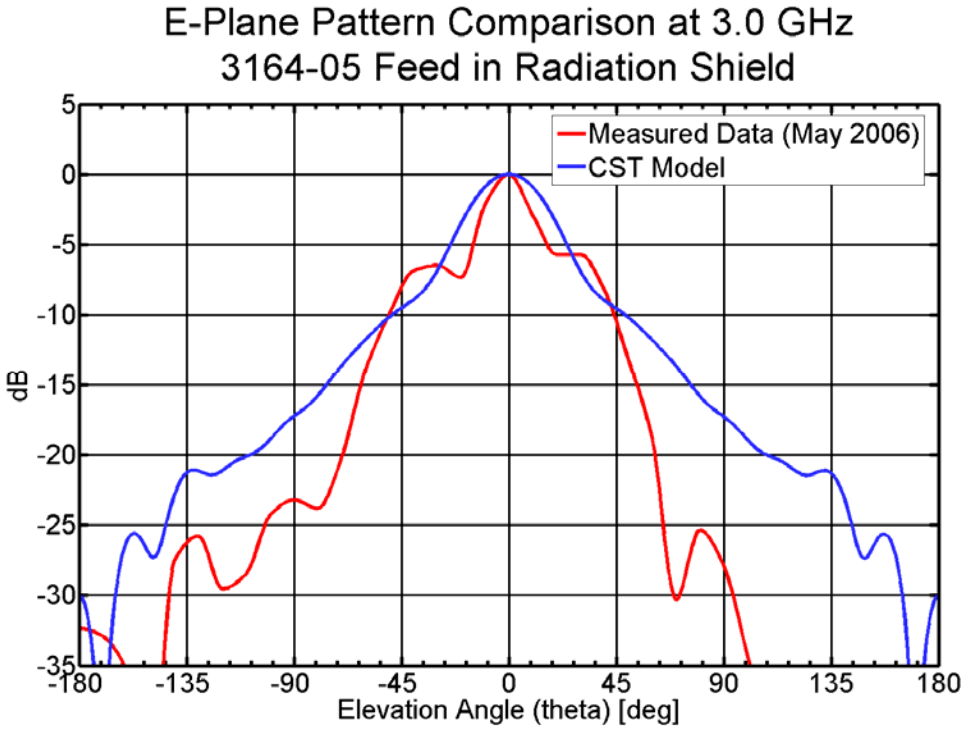


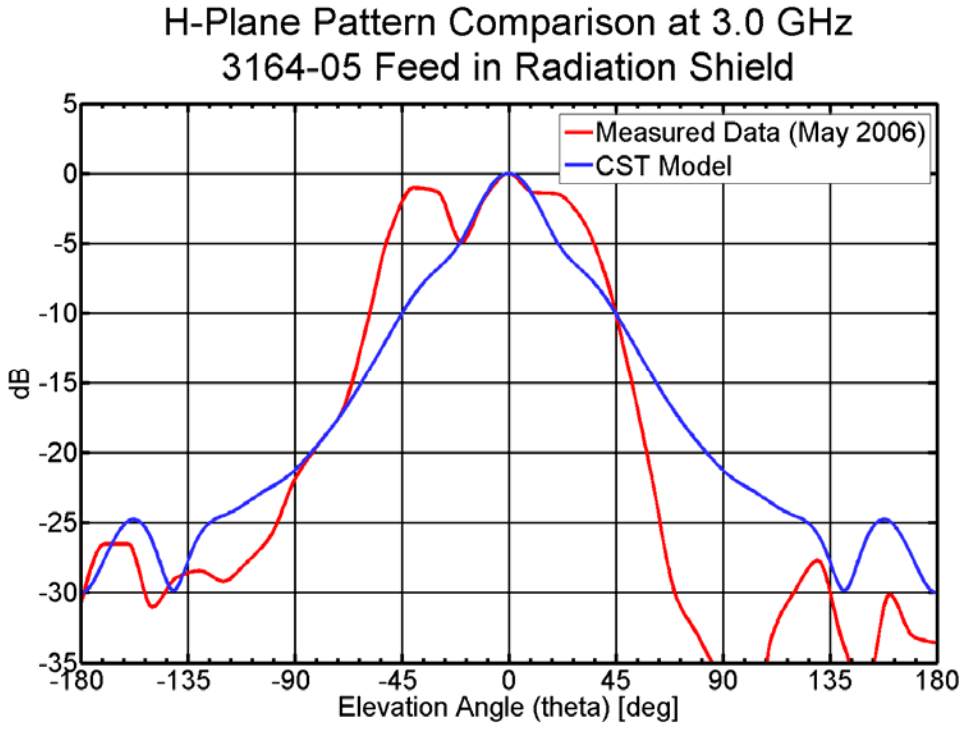
Comparison of Measured Data and CST Model Patterns at 2.2 GHz:



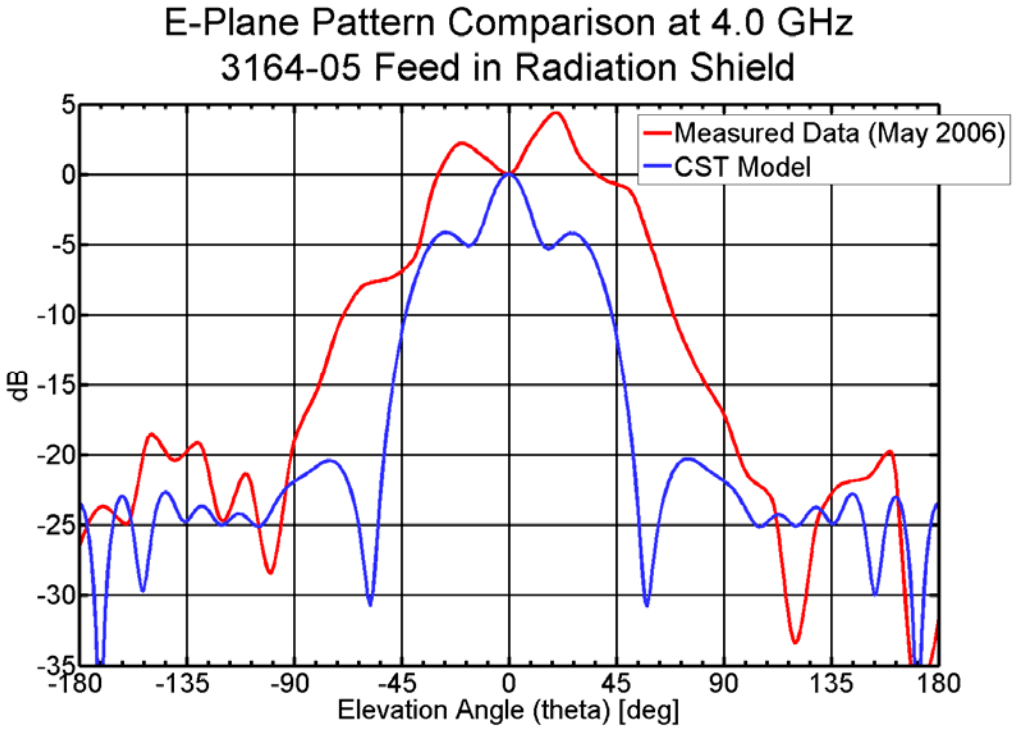


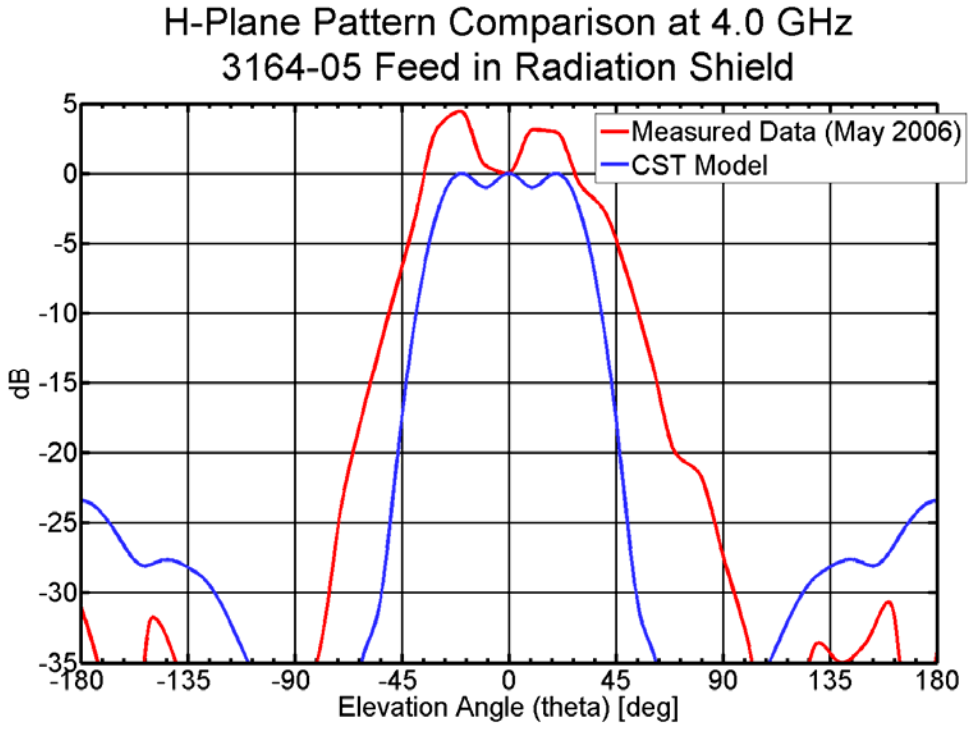
Comparison of Measured Data and CST Model Patterns at 3.0 GHz:



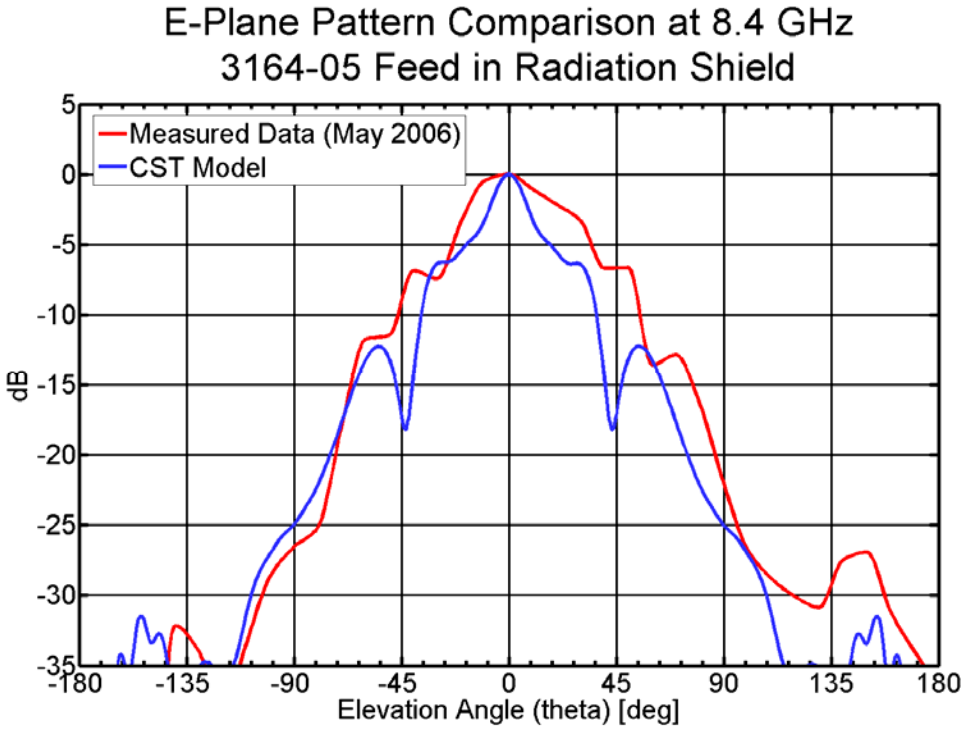


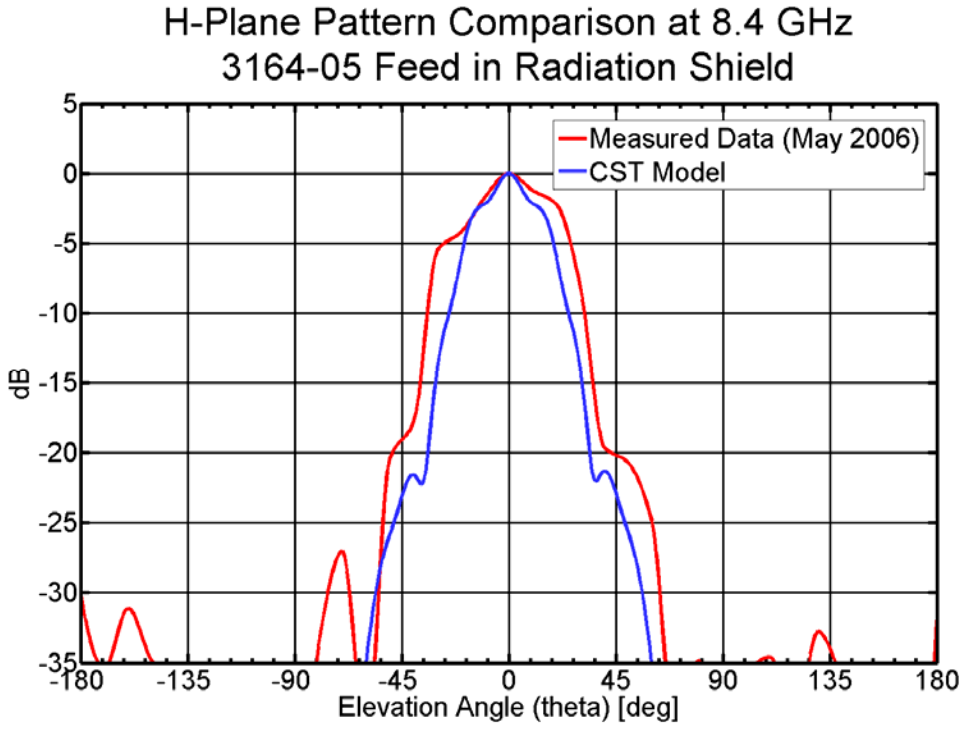
Comparison of Measured Data and CST Model Patterns at 4.0 GHz:



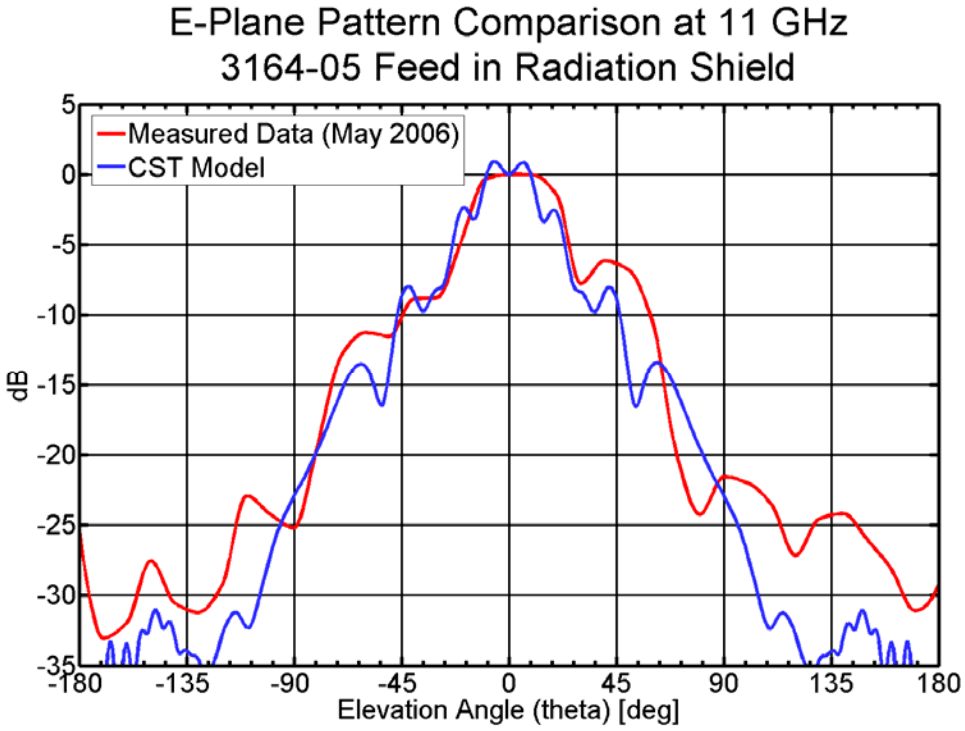


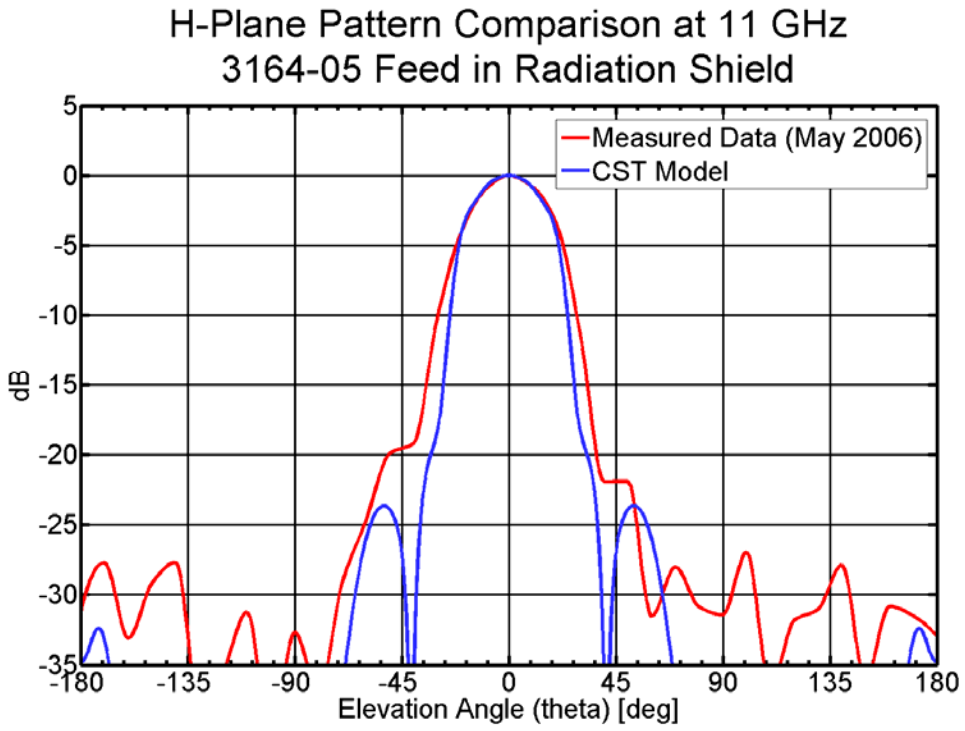
Comparison of Measured Data and CST Model Patterns at 8.4 GHz:



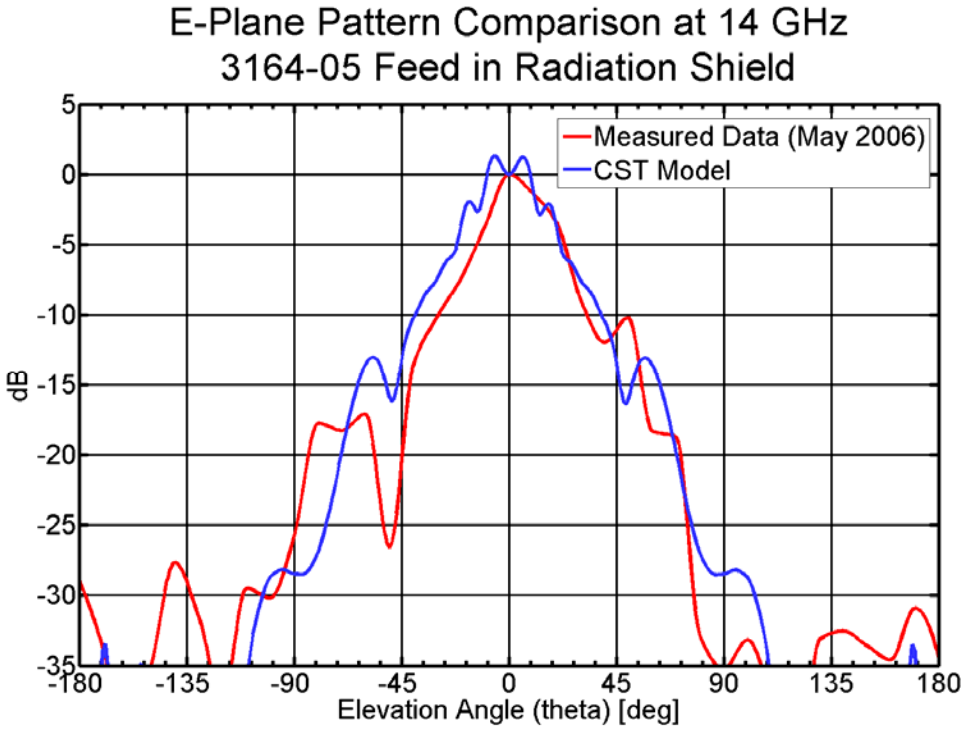


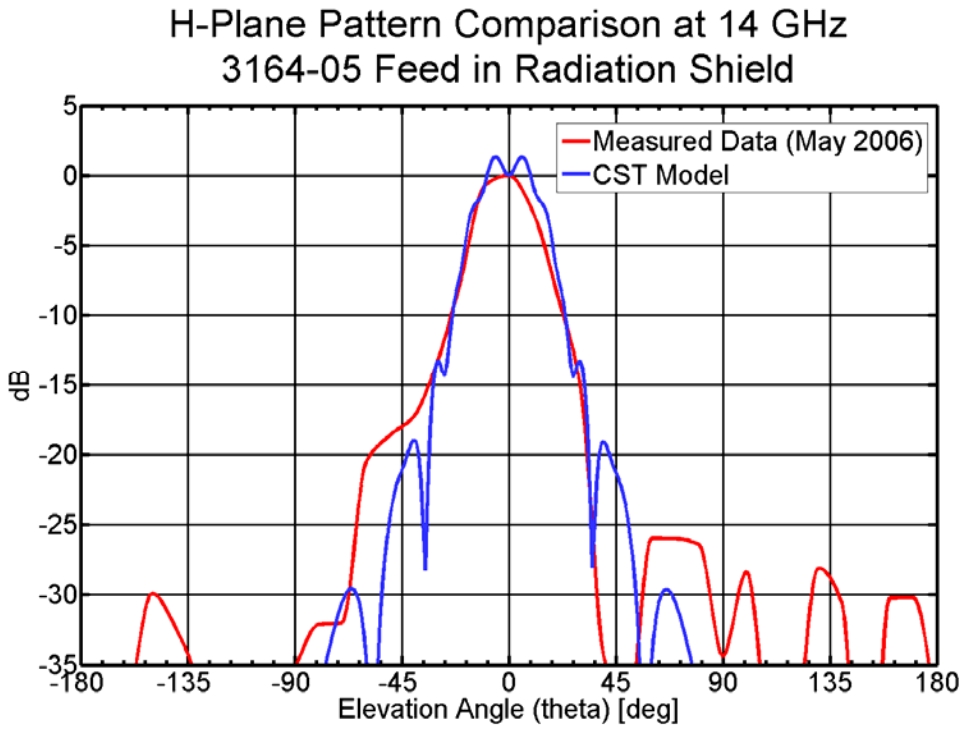
Comparison of Measured Data and CST Model Patterns at 11 GHz:



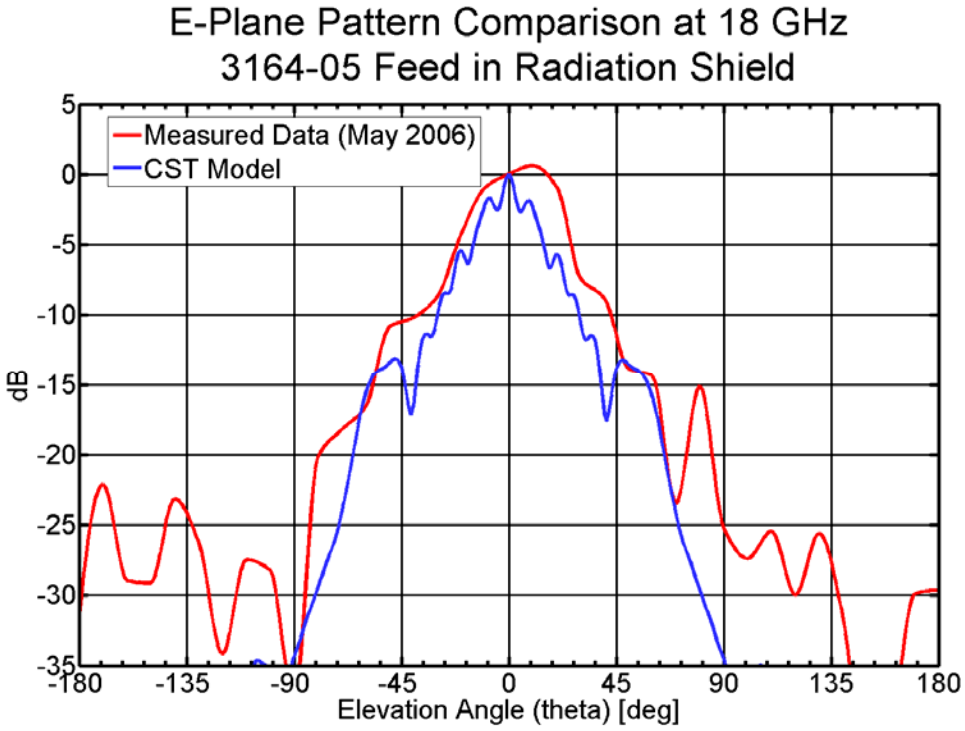


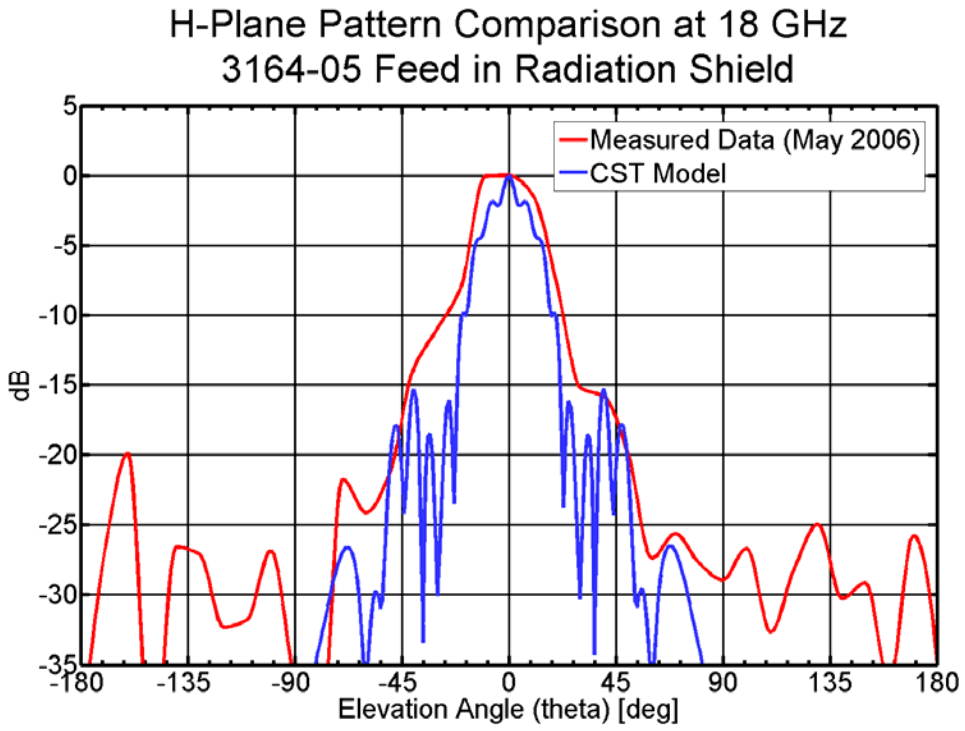
Comparison of Measured Data and CST Model Patterns at 14 GHz:





Comparison of Measured Data and CST Model Patterns at 18 GHz:





6.3.4 Far-Field Patterns for Feed in Absorber Lined Radiation Shield

The images below show the actual feed in the AN-73 absorber lined radiation shield and the CST model.

Lindgren 3164-05 Actual Feed

Figure 6.3.4 a

Lindgren 3164-05 CST Model

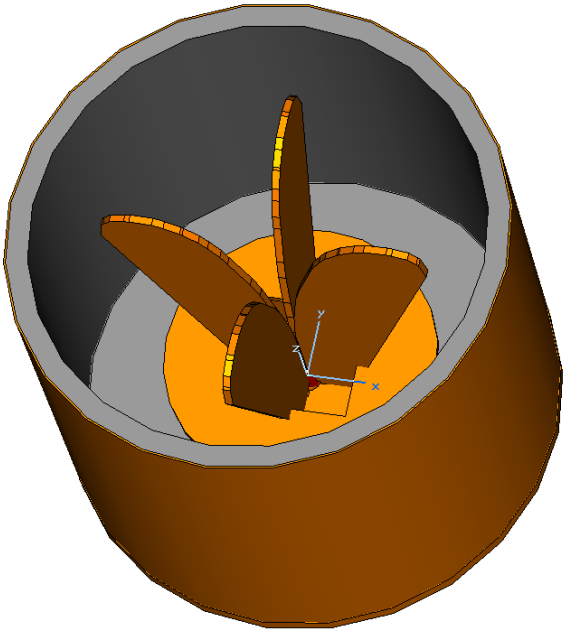
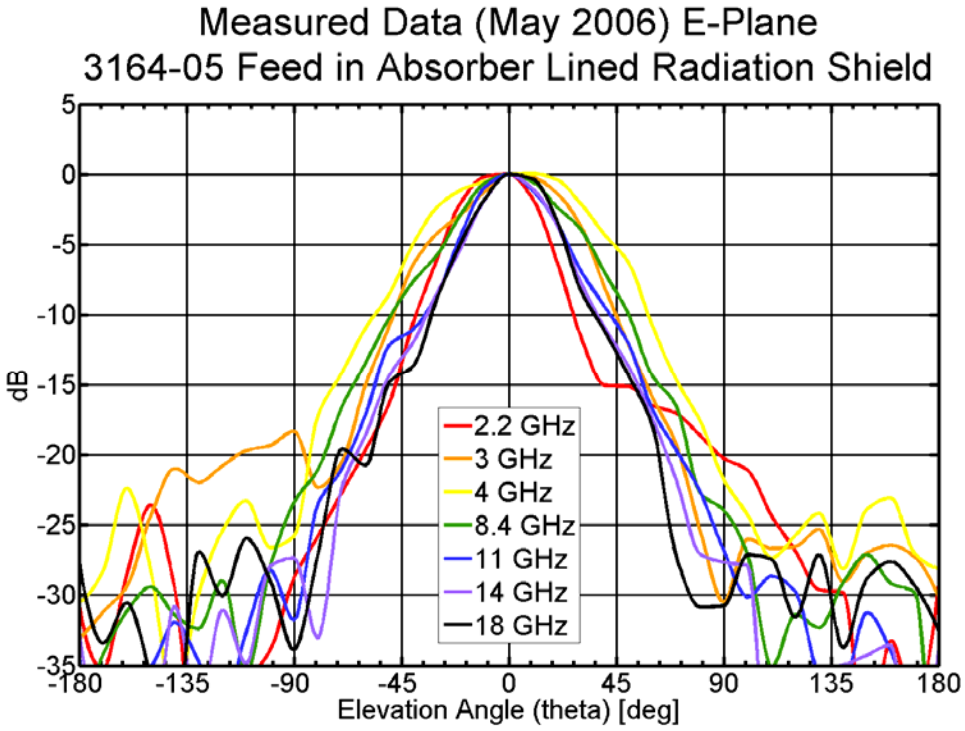
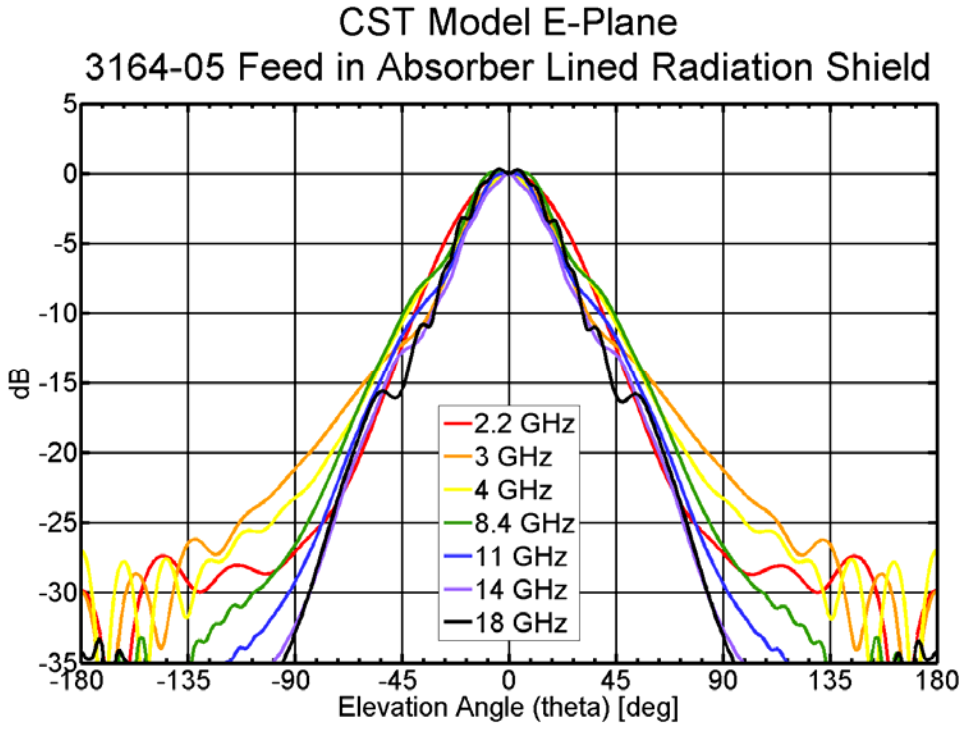


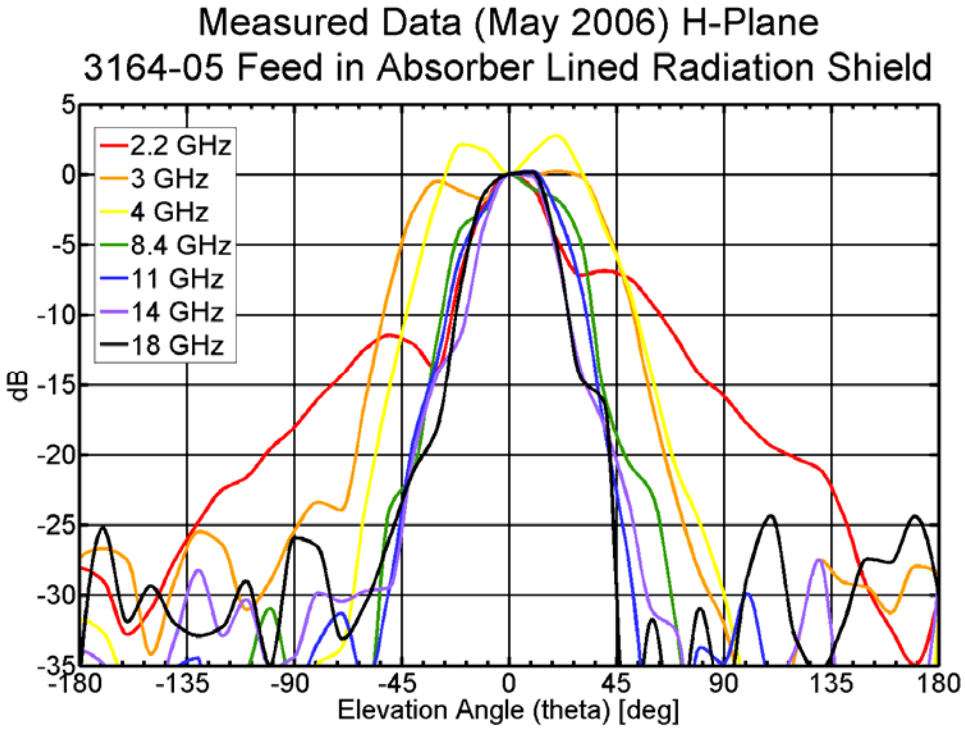
Figure 6.3.4 b

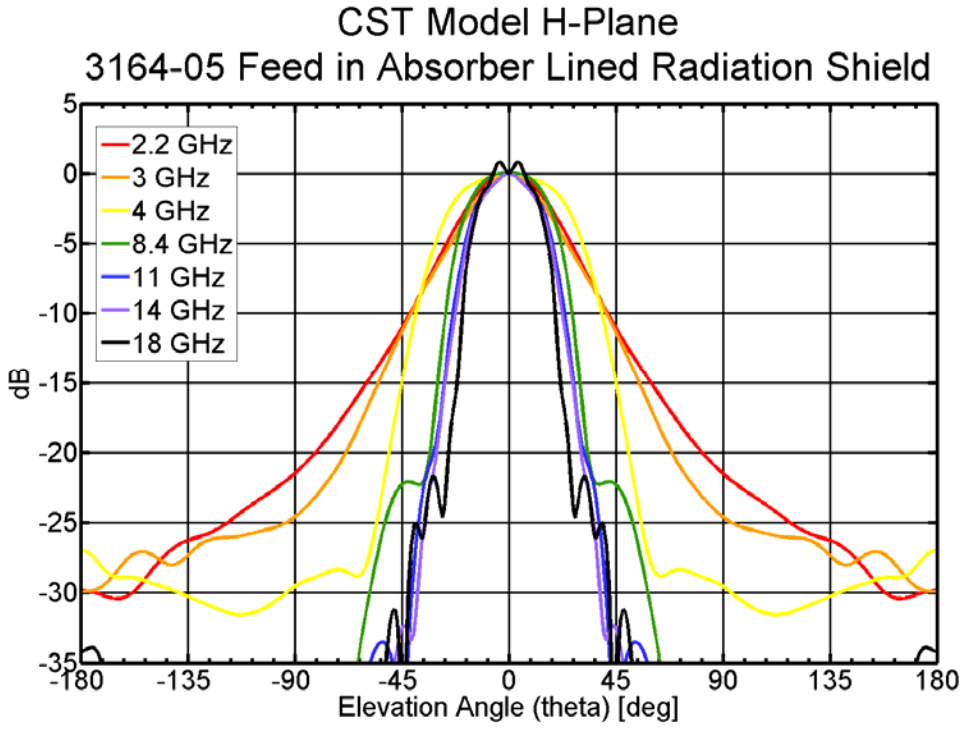
Comparison of Measured Data and CST Model E-Plane Patterns:



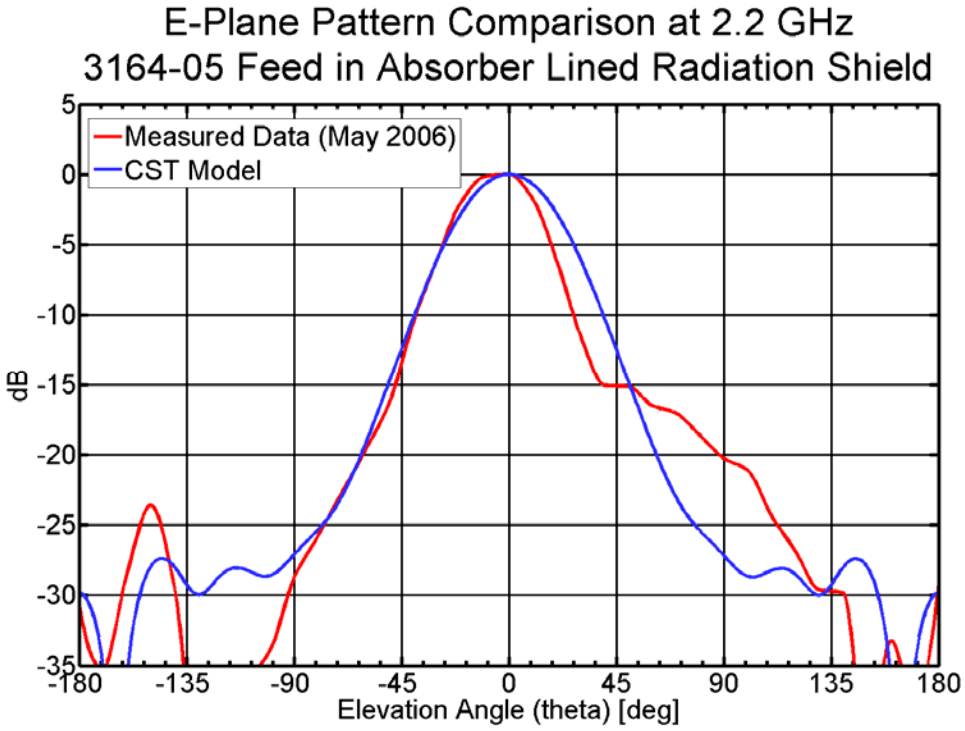


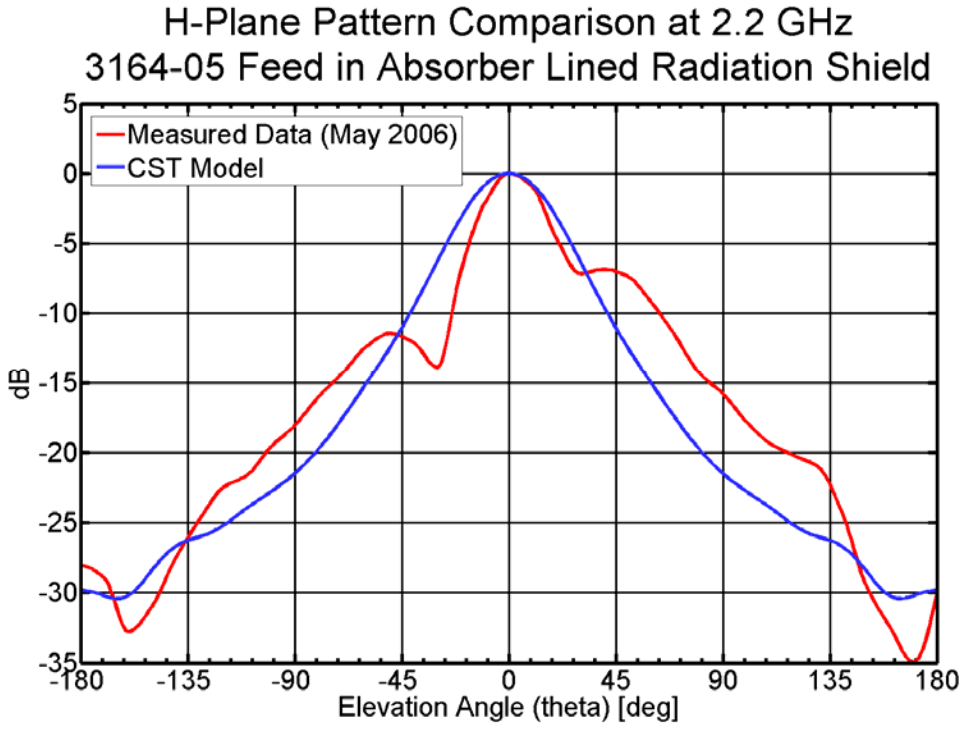
Comparison of Measured Data and CST Model H-Plane Patterns:



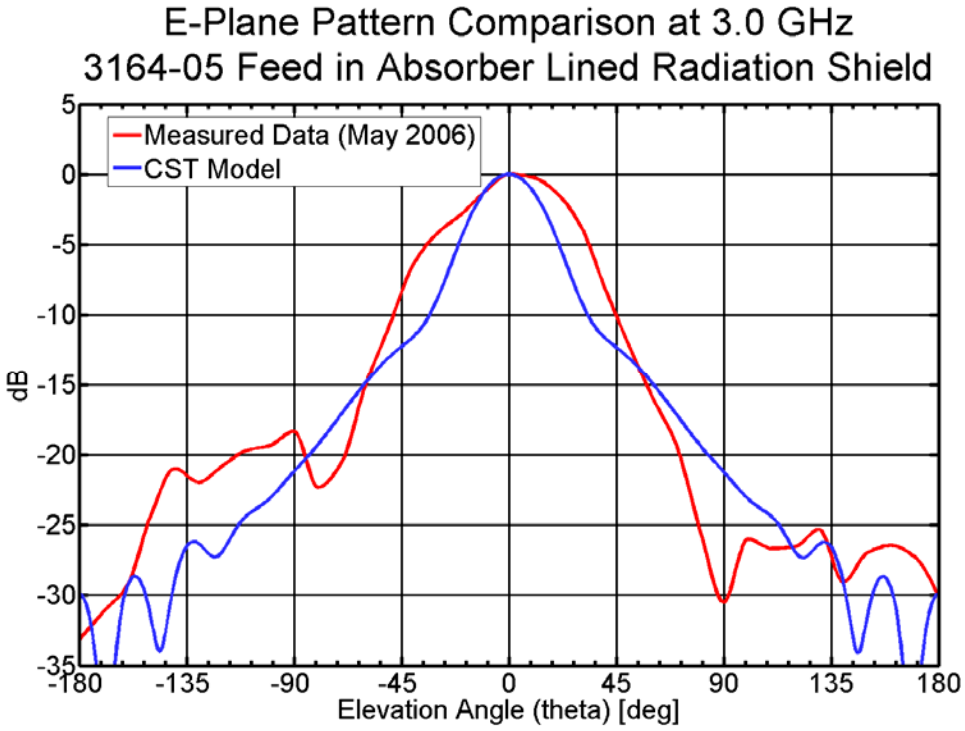


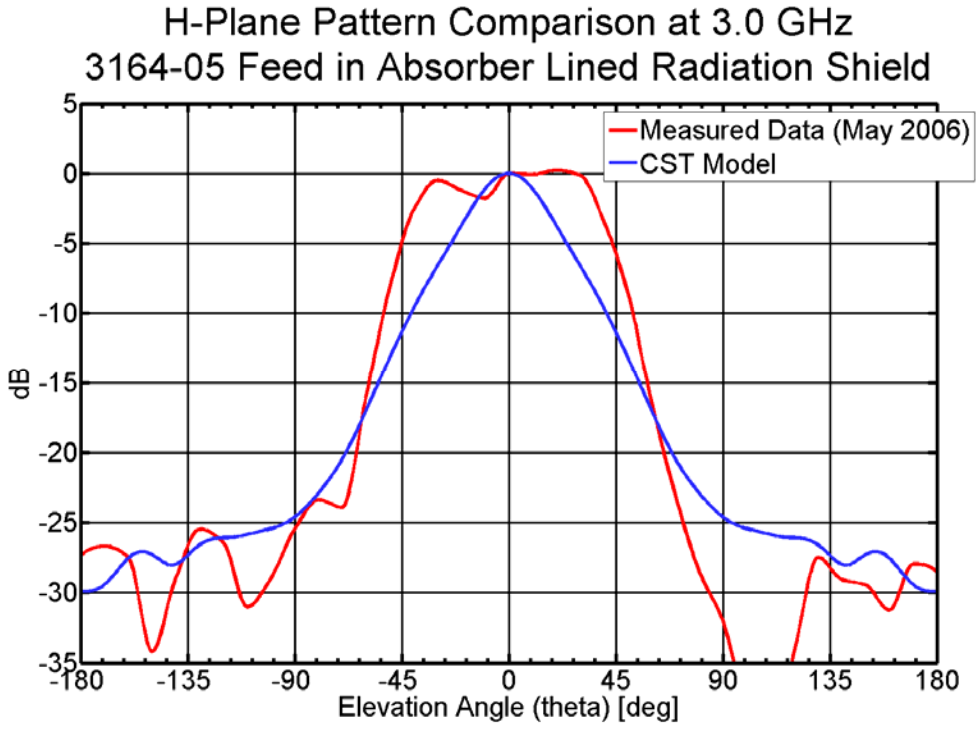
Comparison of Measured Data and CST Model Patterns at 2.2 GHz:



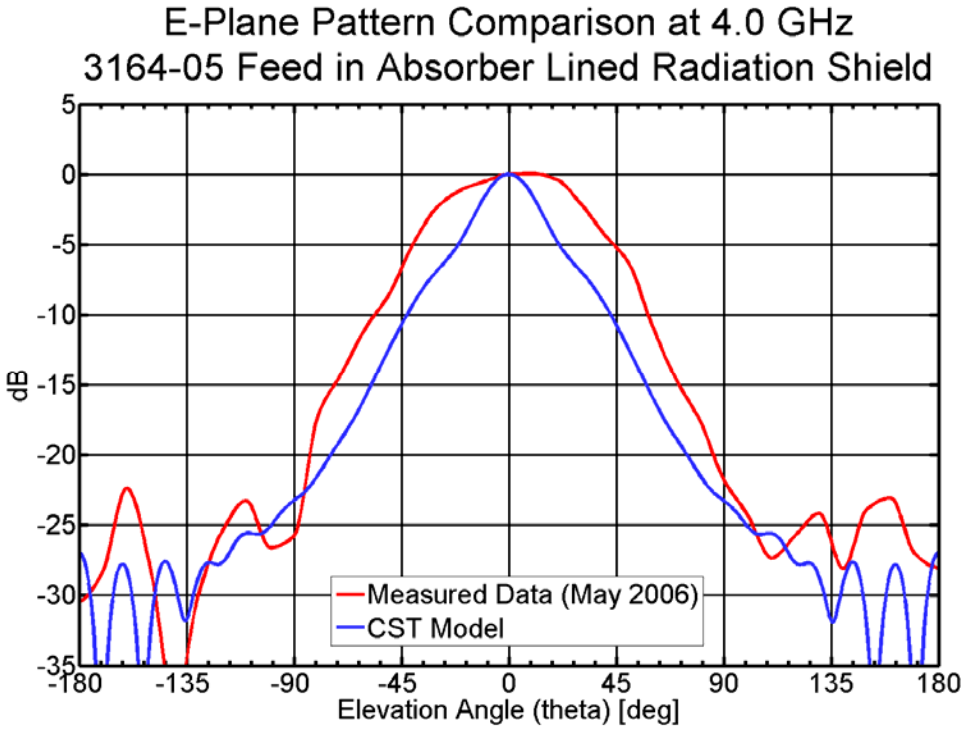


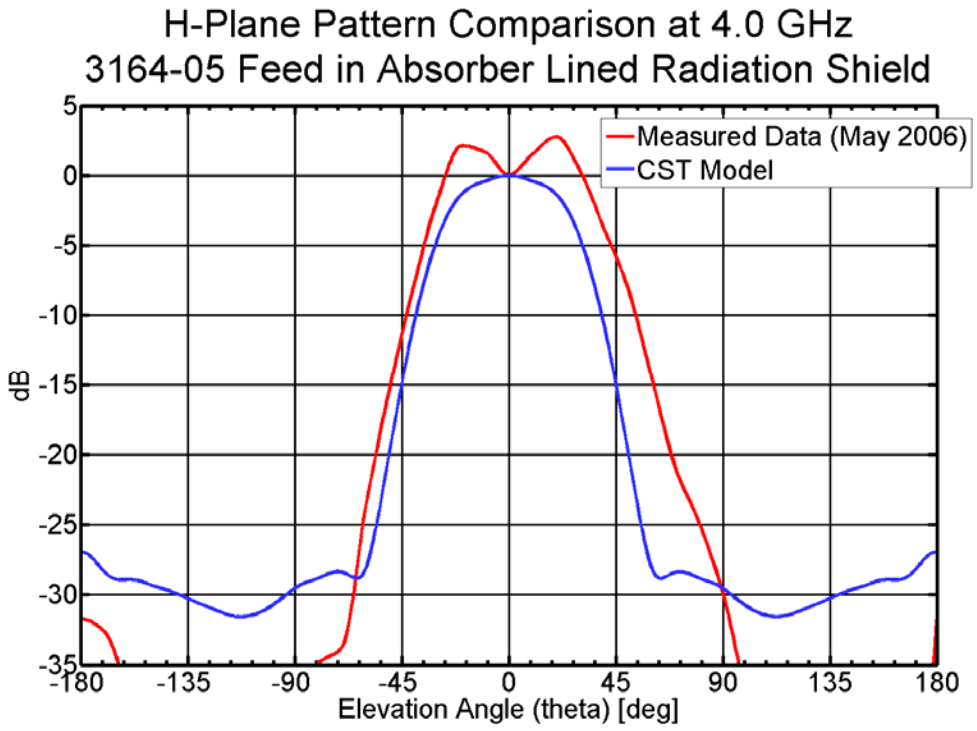
Comparison of Measured Data and CST Model Patterns at 3.0 GHz:



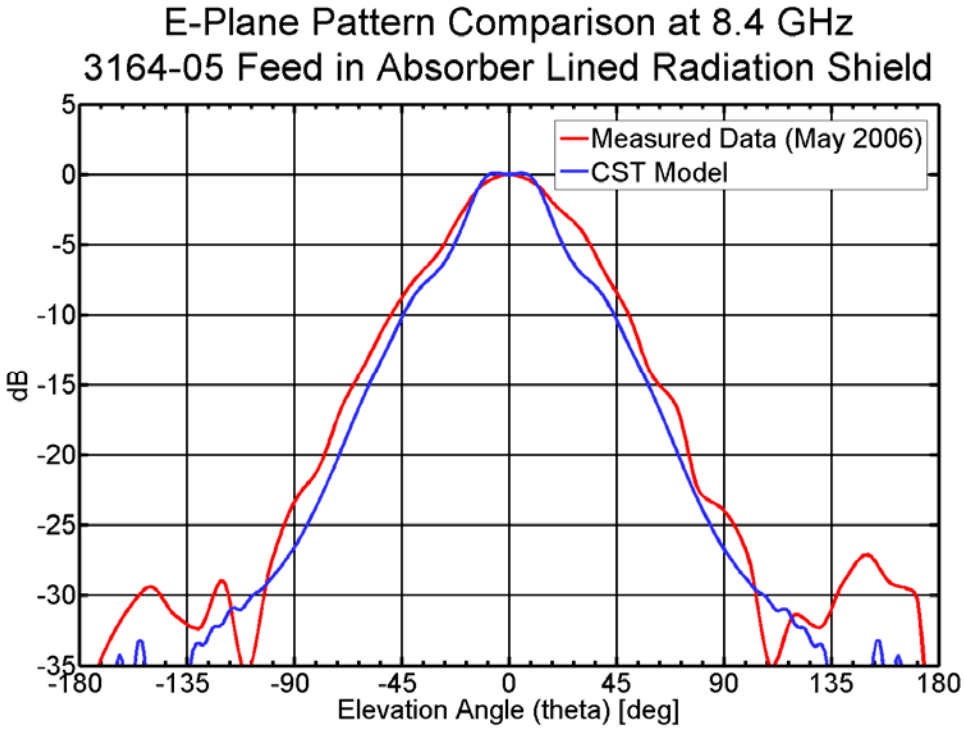


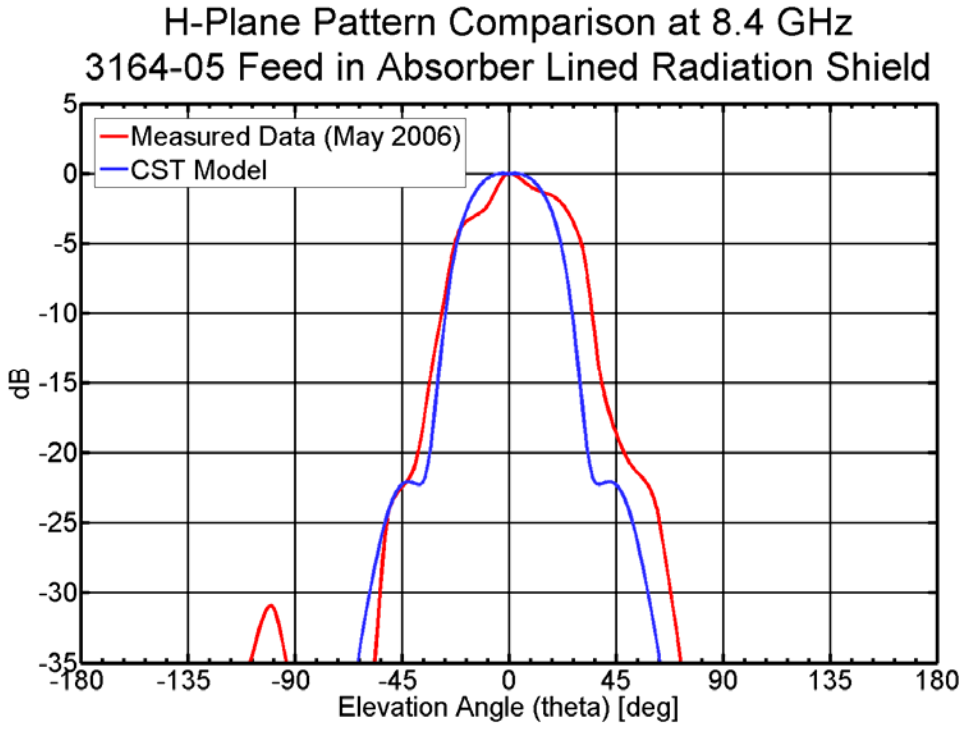
Comparison of Measured Data and CST Model Patterns at 4.0 GHz:



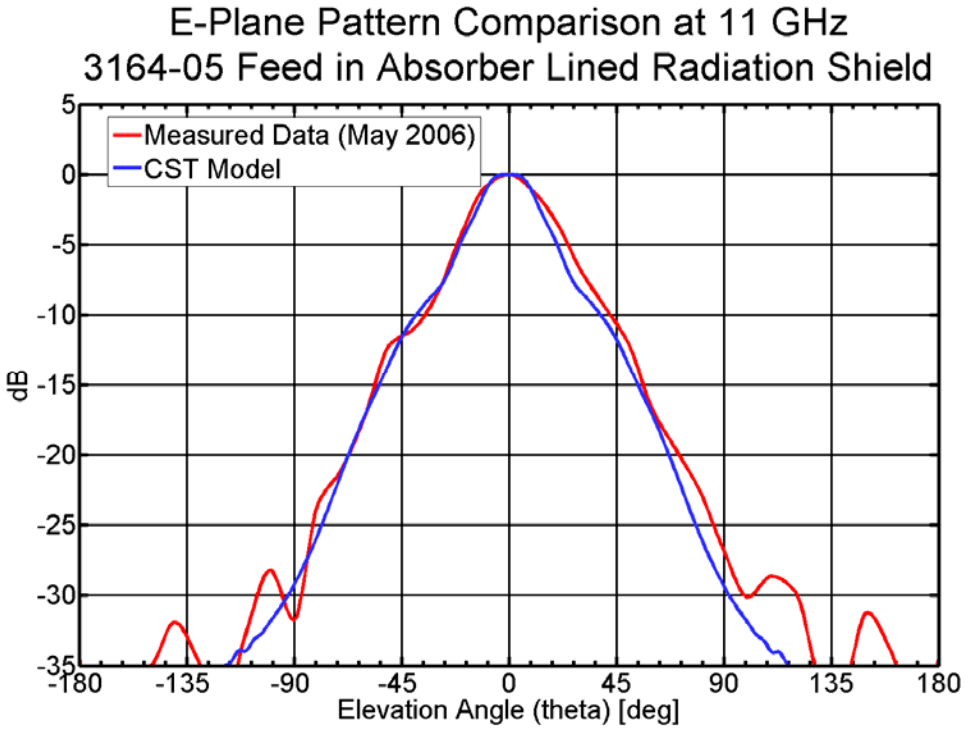


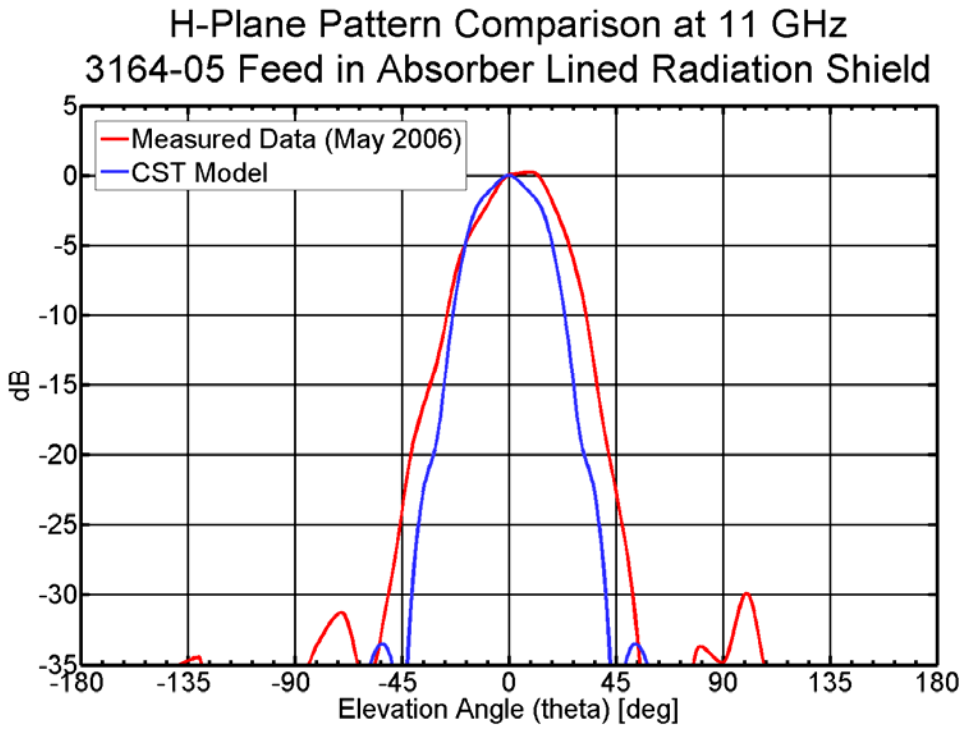
Comparison of Measured Data and CST Model Patterns at 8.4 GHz:



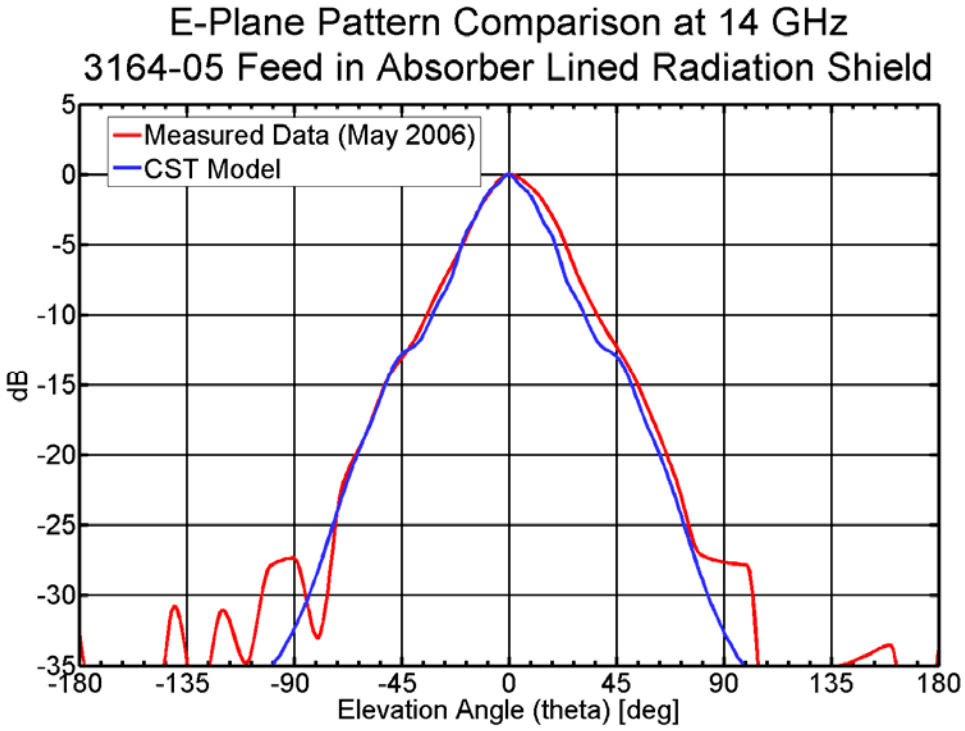


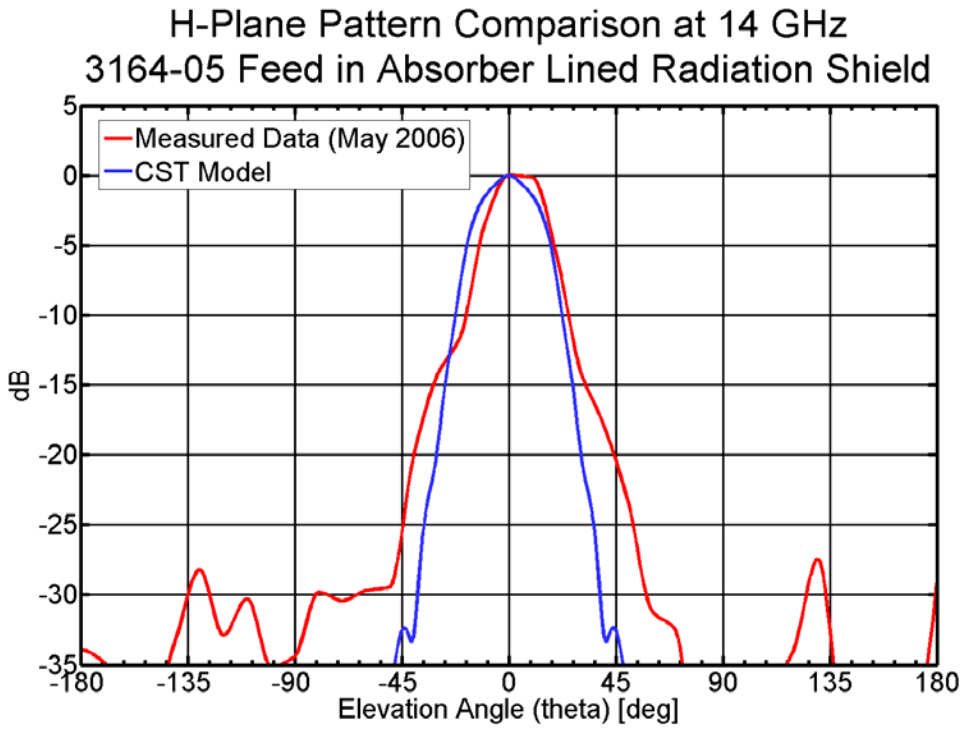
Comparison of Measured Data and CST Model Patterns at 11 GHz:



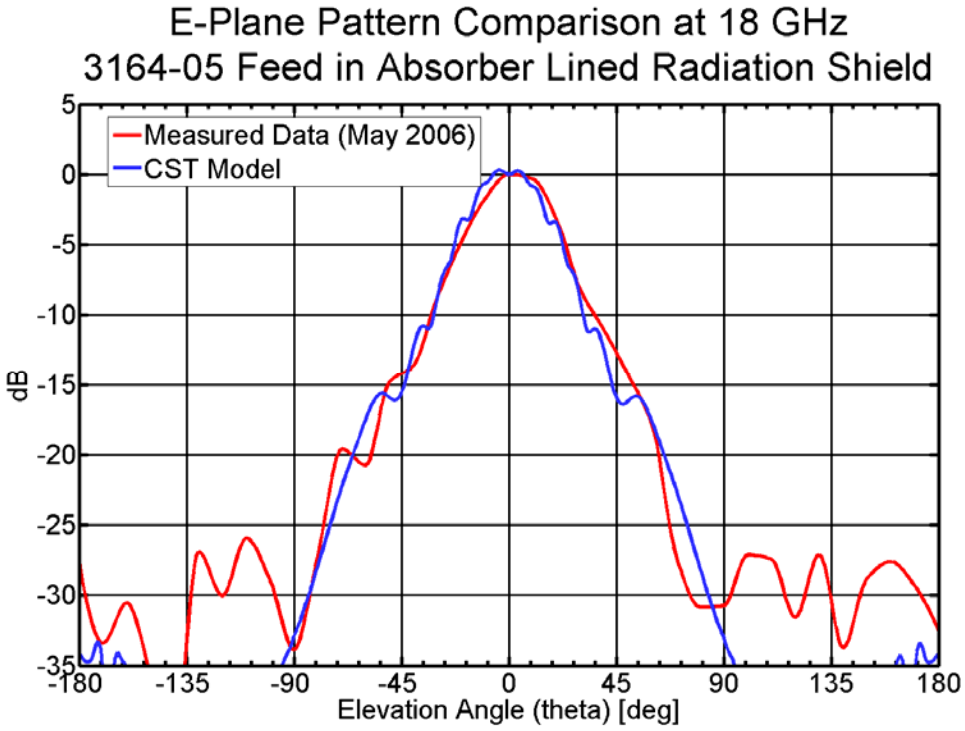


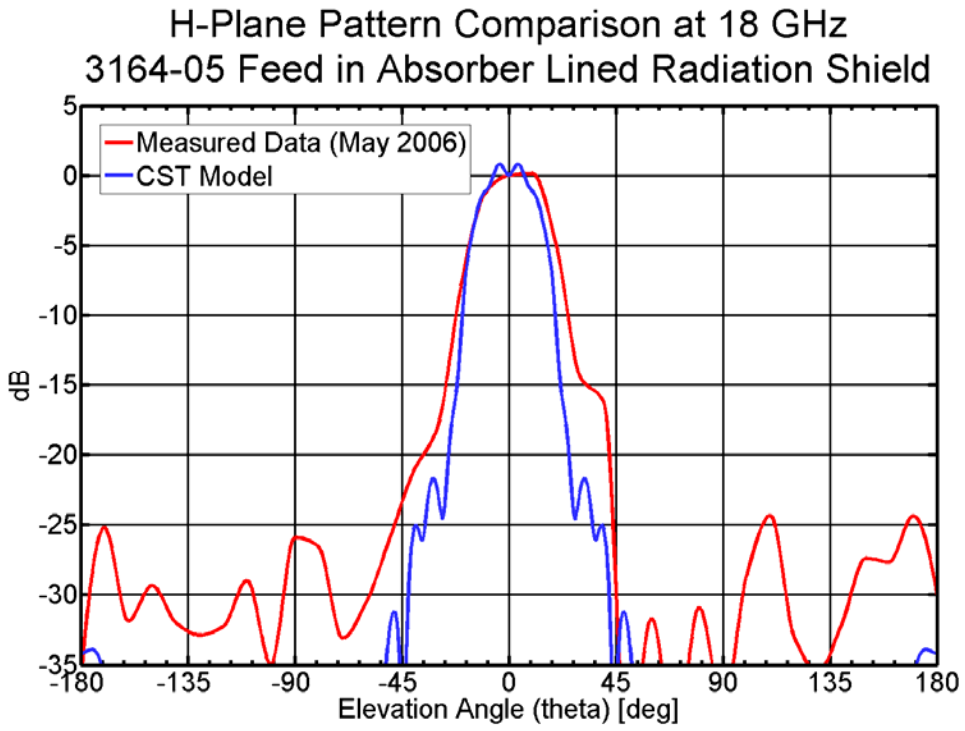
Comparison of Measured Data and CST Model Patterns at 14 GHz:





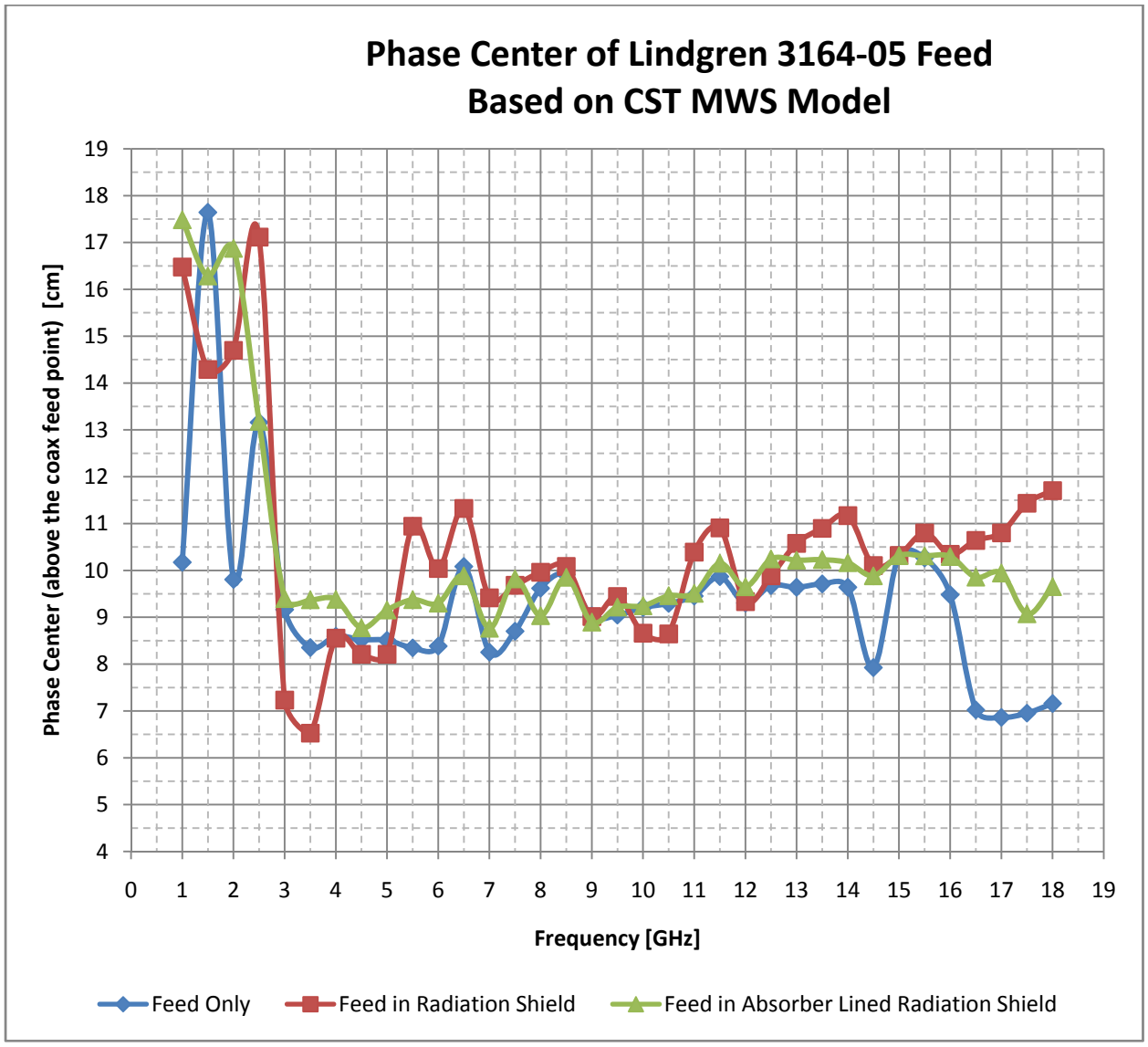
Comparison of Measured Data and CST Model Patterns at 18 GHz:



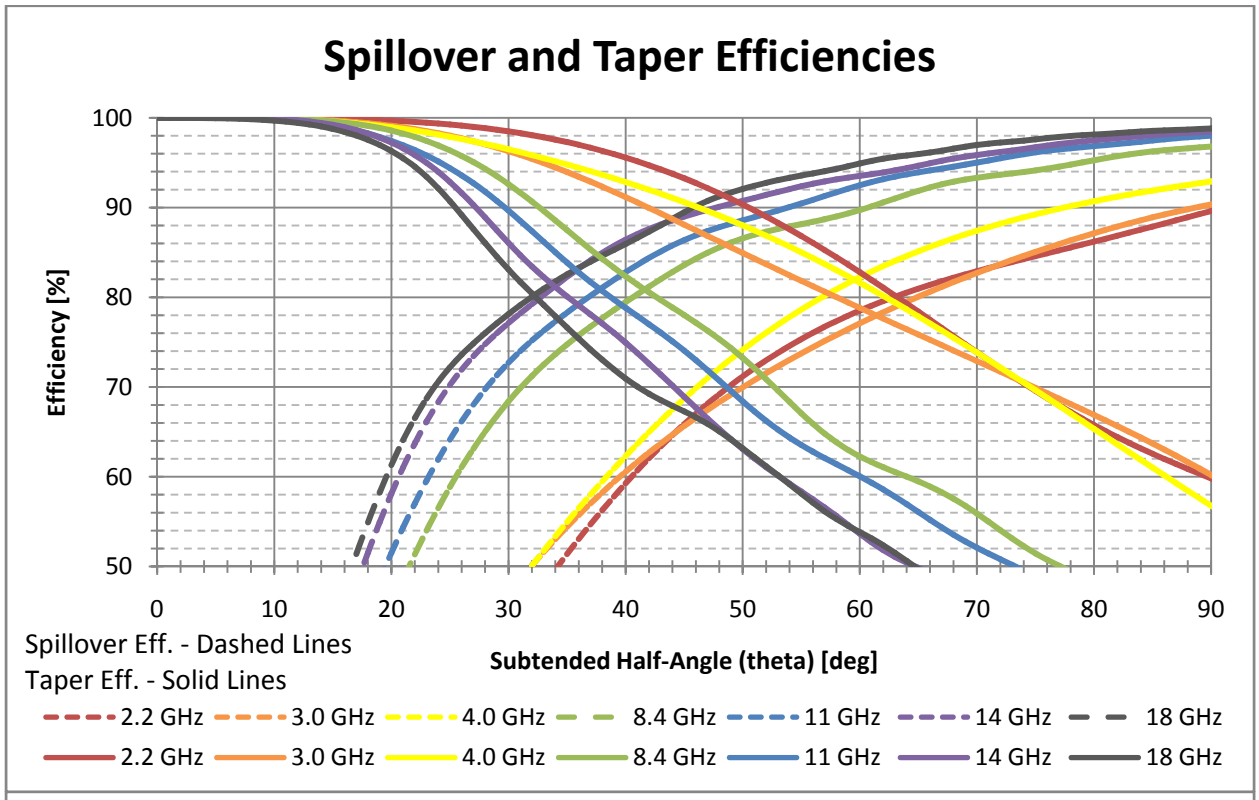


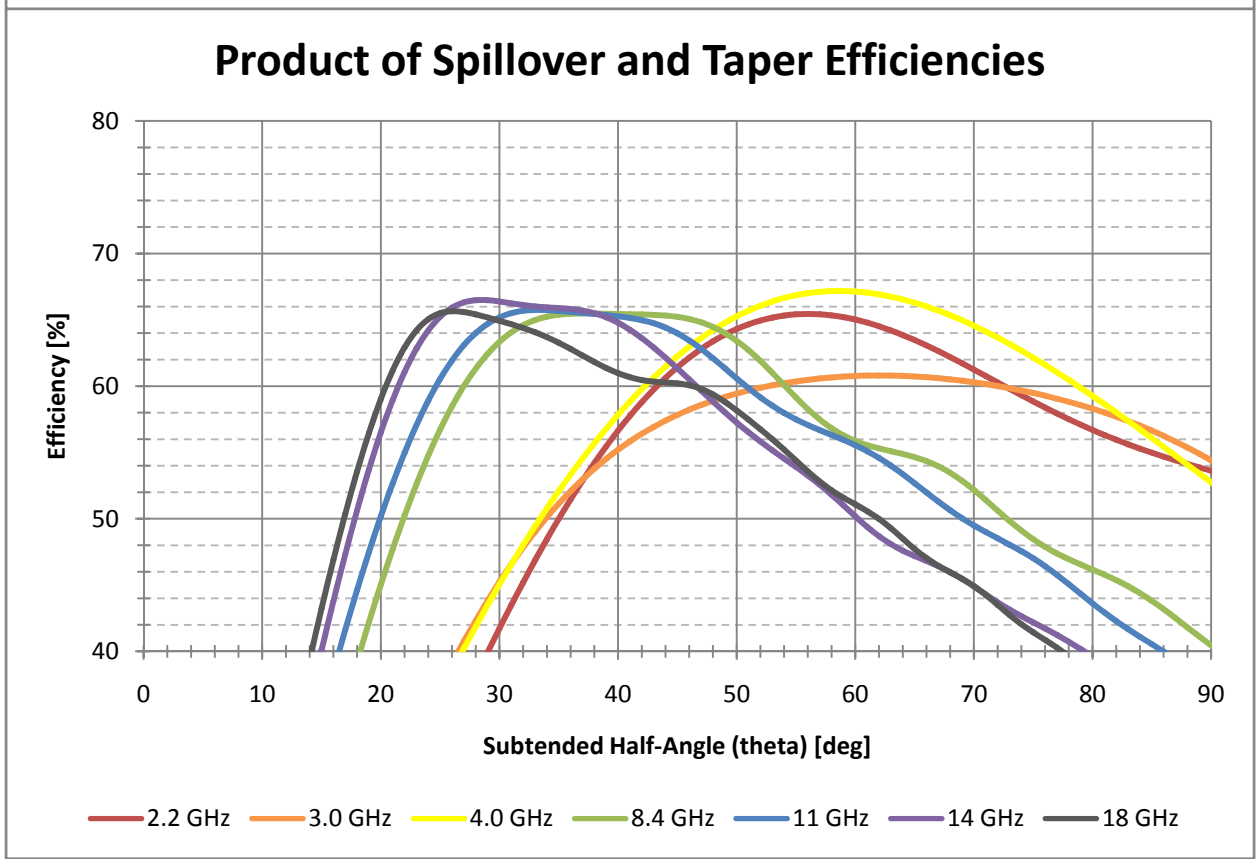
6.3.5 Phase Center

The phase center variation appears to be best using the absorber lined radiation shield.

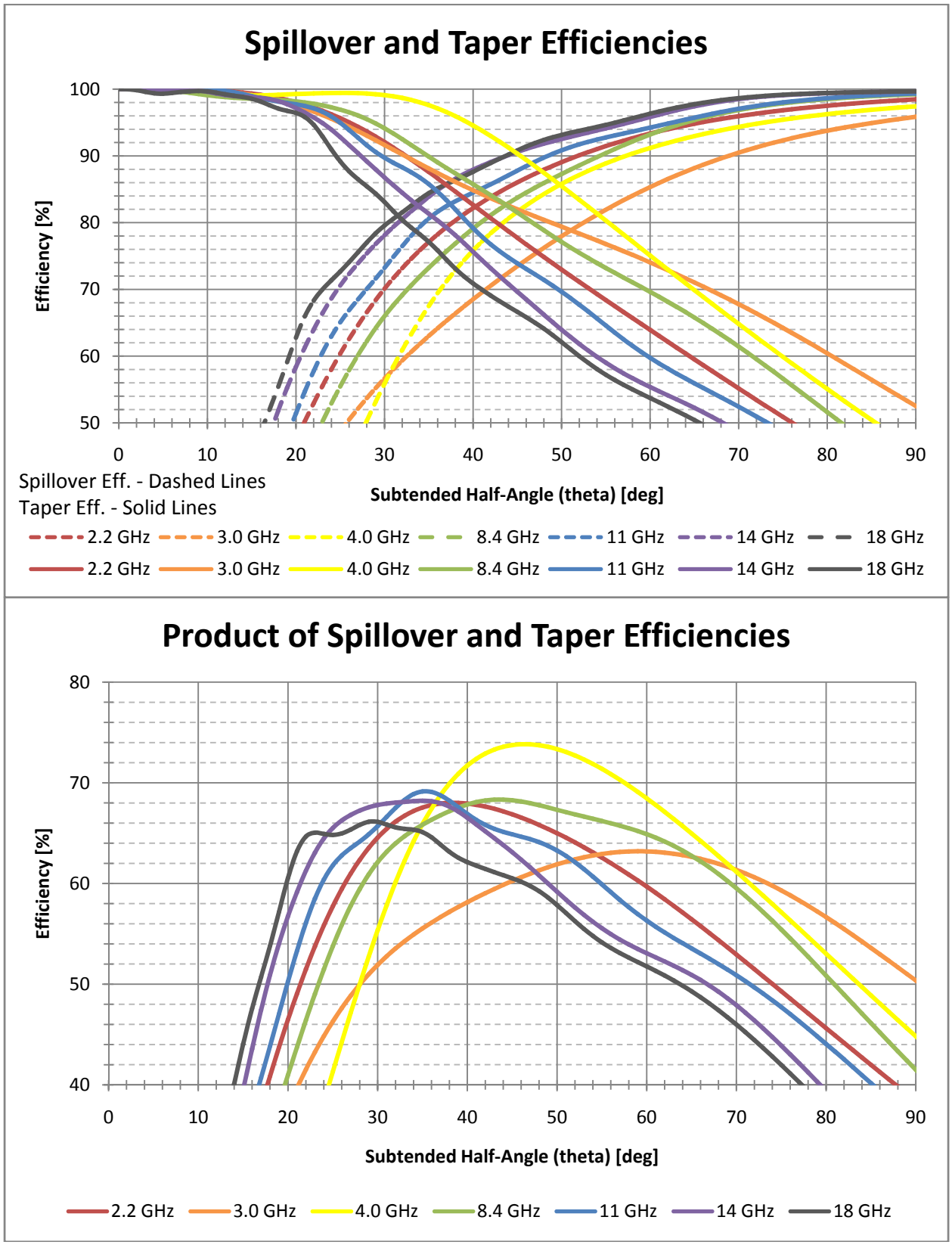

Phase Center of Lindgren 3164-05 Feed
Based on CST MWS Model
Phase Center (above the coax feed point) [cm]
Frequency [GHz]
Feed Only
Feed in Radiation Shield
Feed in Absorber Lined Radiation Shield

6.3.6 Efficiencies for Feed Only

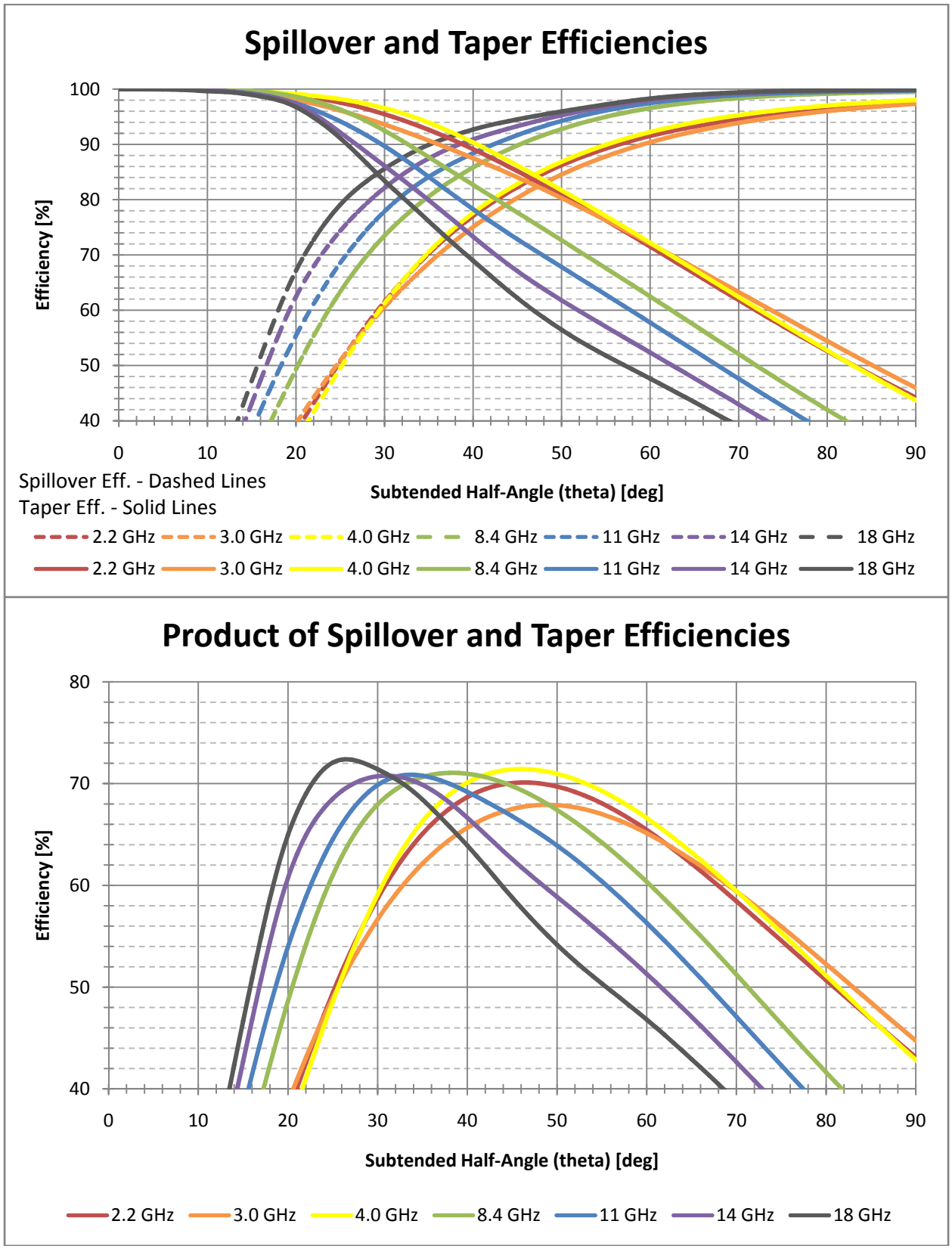




6.3.7 Efficiencies for Feed in Radiation Shield



6.3.8 Efficiencies for Feed in Absorber Lined Radiation Shield



6.4 Data from CST Model of 3164-06 Design (0.3-6 GHz)

The design of the ETS-Lindgren 3164-06 (see section 3.1.3) was modeled and simulated using CST MWS. The following data shows results from this simulation. The images below show the actual feed and the CST model side by side.

Lindgren 3164-06 Actual Feed

Figure 6.4 a

Lindgren 3164-06 CST Model

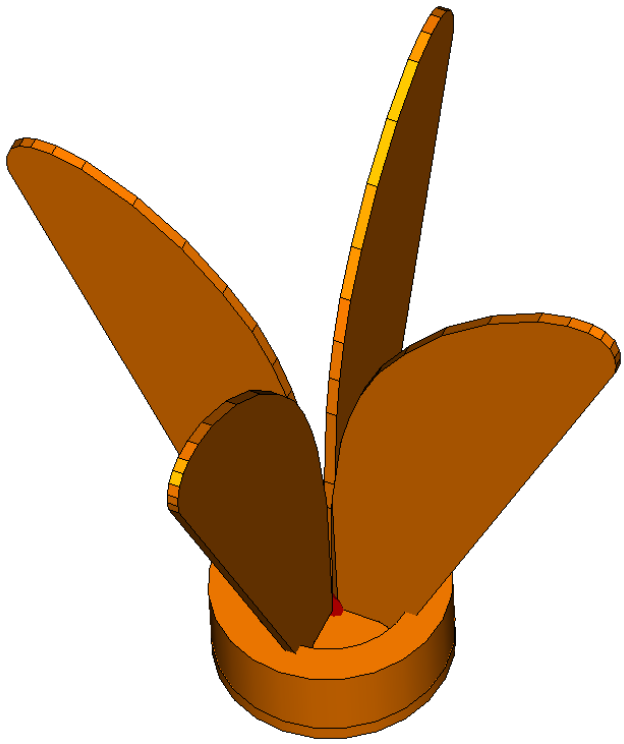
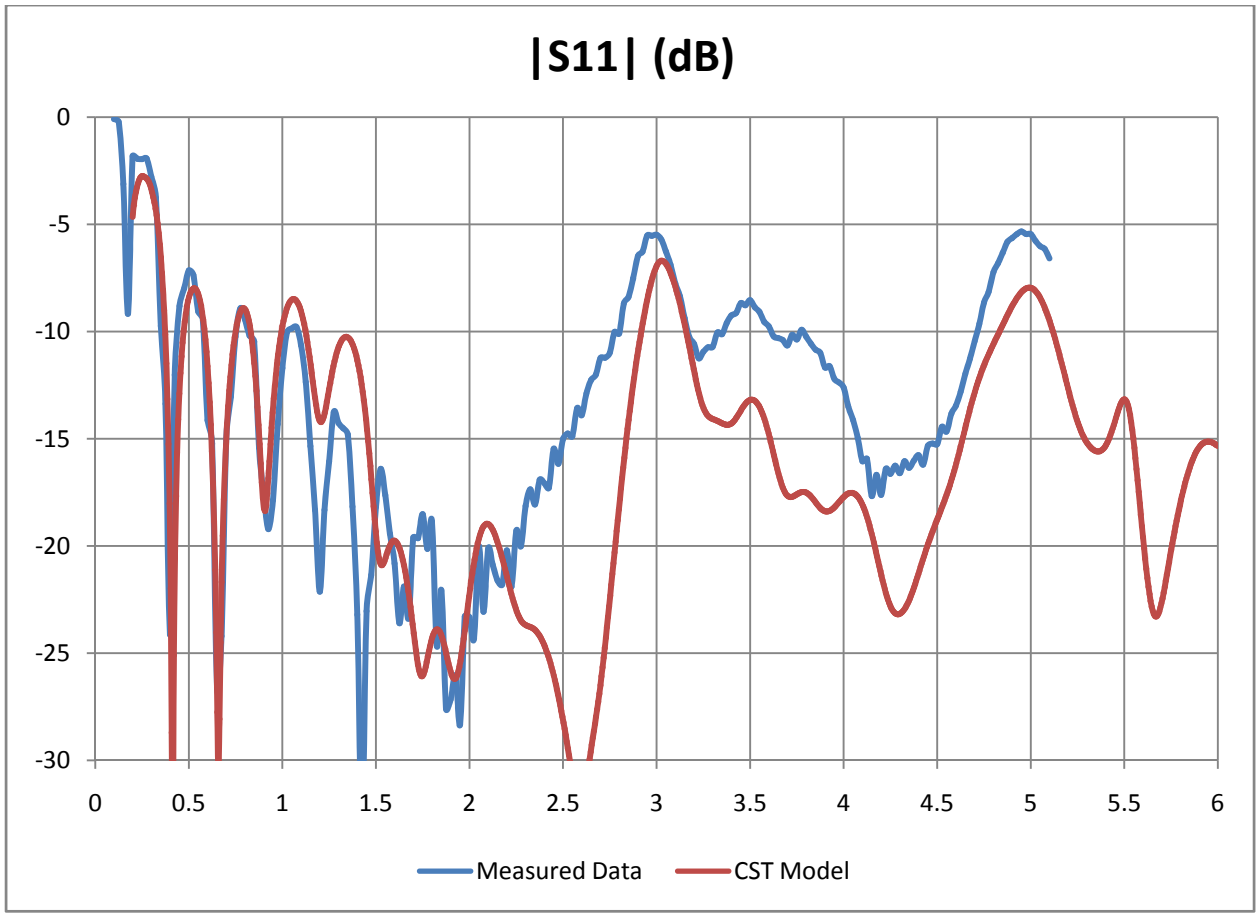


Figure 6.4 b

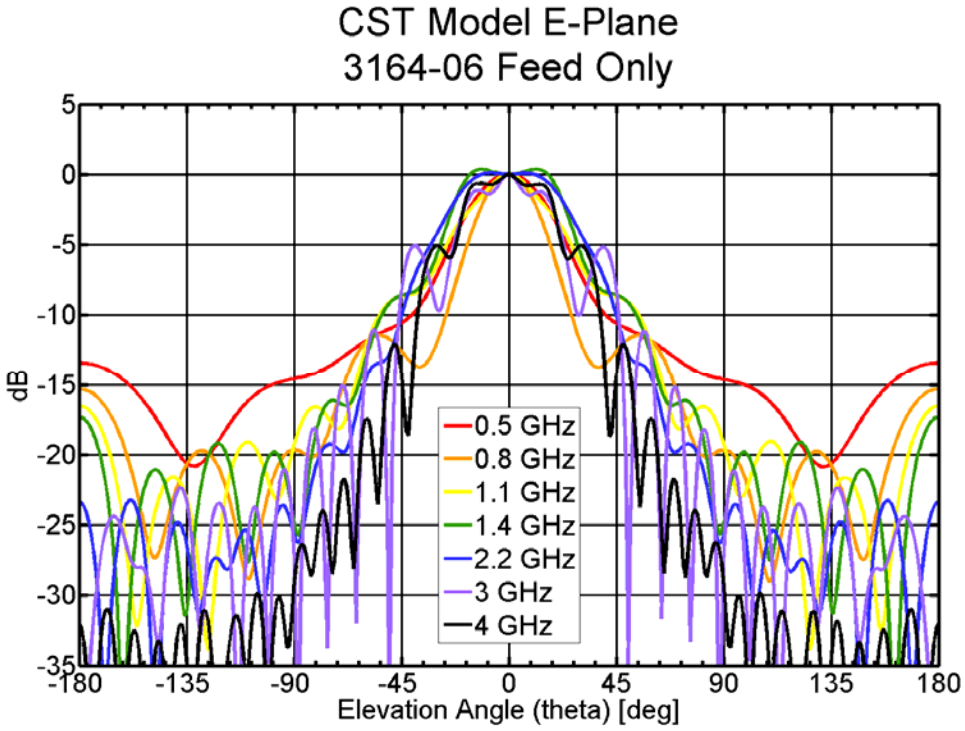
6.4.1 Return Loss

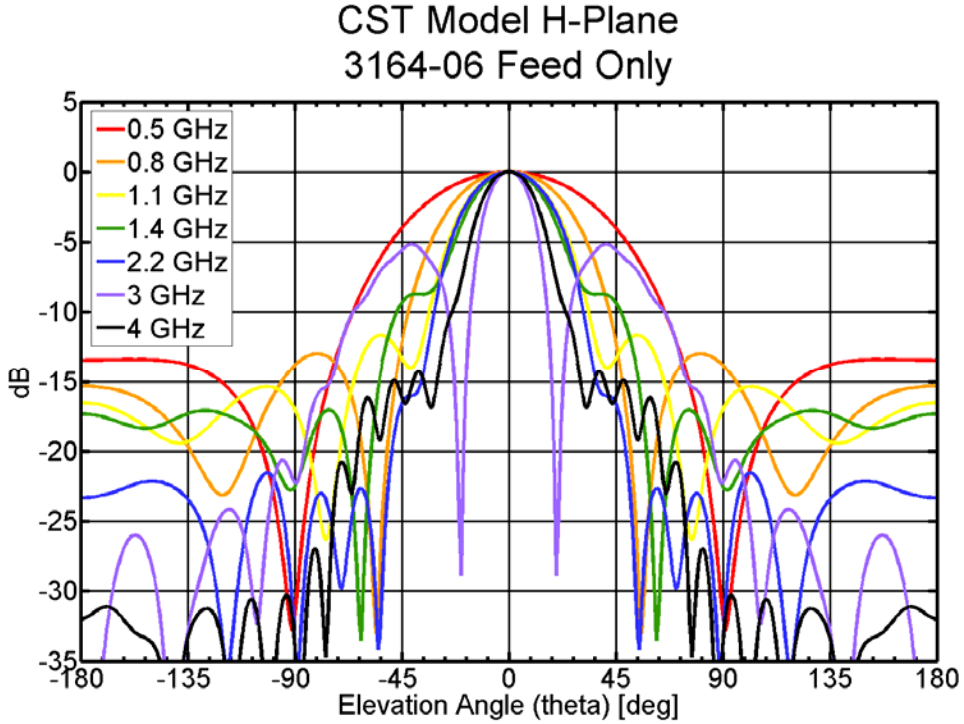
It is important to realize the differences between the model and the actual feed. The input of the actual feed (from which the measured data was taken) is a SMA connector that connects to coax going to a coax-to-slotline transition inside the slot between the fins. The CST model does not have this complicated input network. The model is fed by a simple E-field excitation across the slot at the same point that the coax excitation would be located. This difference should explain significant variations between the modeled return loss and the measured data. The measured data is of the V-polarization with the lossy backshort removed. The CST model is of the same structure.


|S11| (dB)
0
-5
-10
-15
-20
-25
-30
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
Measured Data
CST Model

6.4.2 Far-Field Patterns for Feed Only

Principal Planes of the CST Model Patterns:





6.4.3 Far-Field Patterns for Feed in Radiation Shield

The image below shows CST model of the Lindgren 3164-06 feed inside of a radiation shield.

Lindgren 3164-06 CST Model

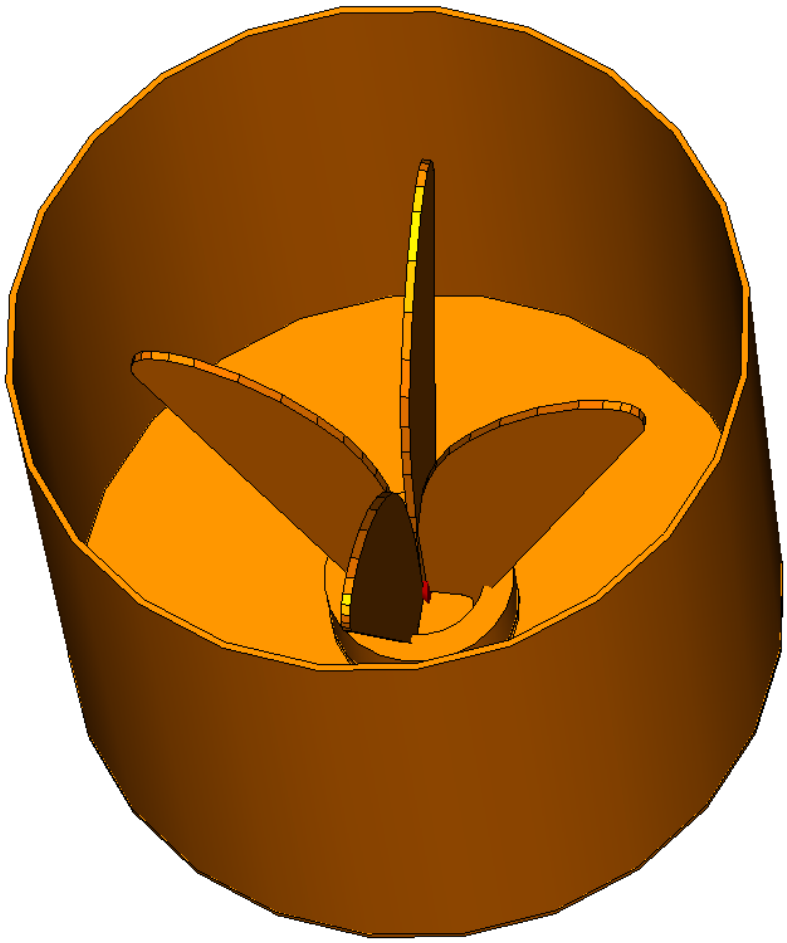
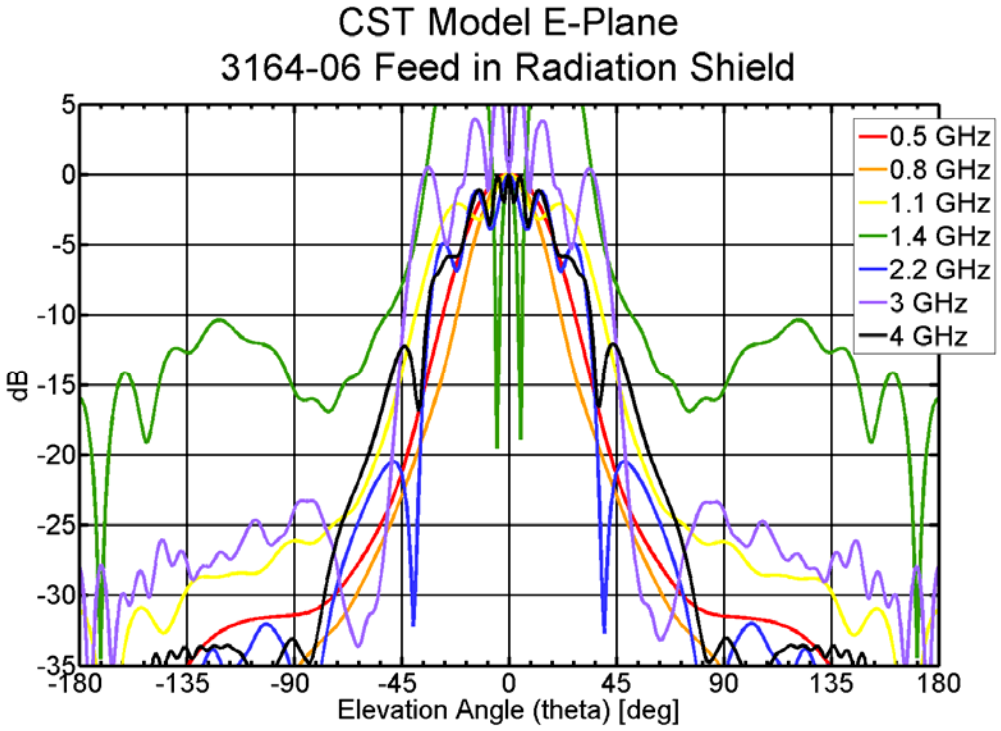
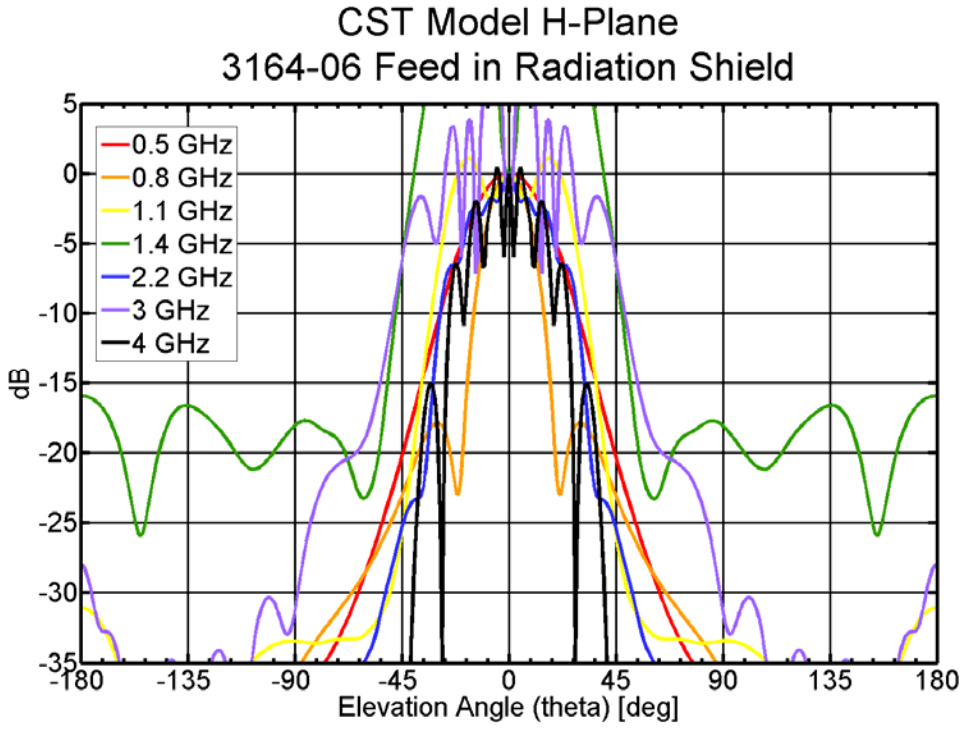


Figure 6.4.3 a

Principal Planes of the CST Model Patterns:





6.4.4 Far-Field Patterns for Feed in Absorber Lined Radiation Shield

The image below shows CST model of the Lindgren 3164-06 feed inside of an absorber lined radiation shield. Note that this absorber model is supposed to match the properties of AN-73, which should not act as a good absorber at the operating frequencies of this feed.

Lindgren 3164-06 CST Model

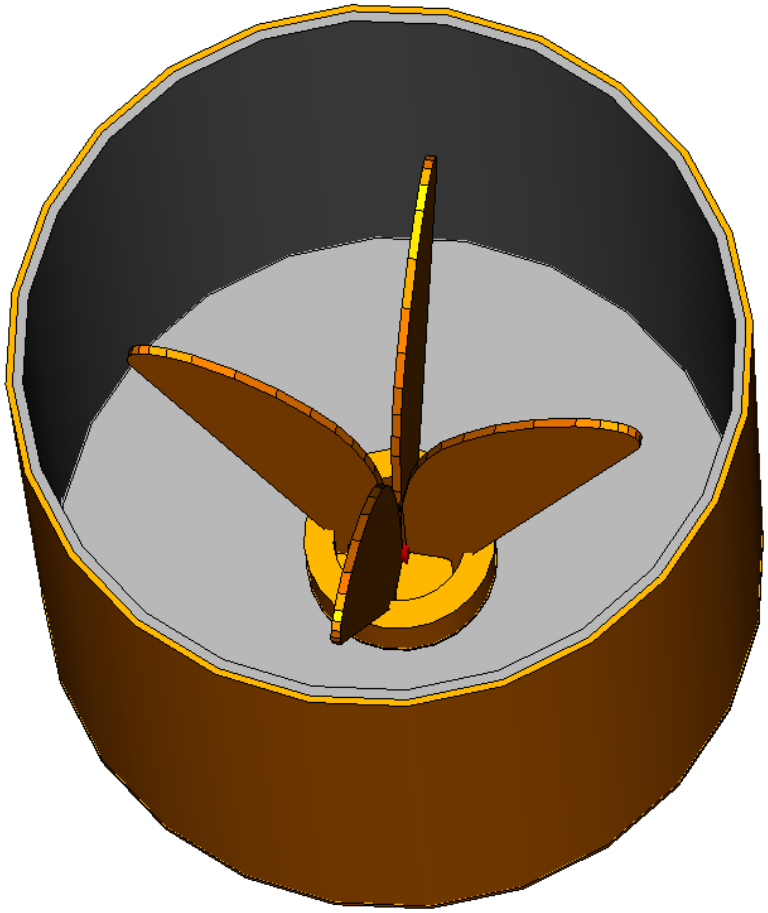
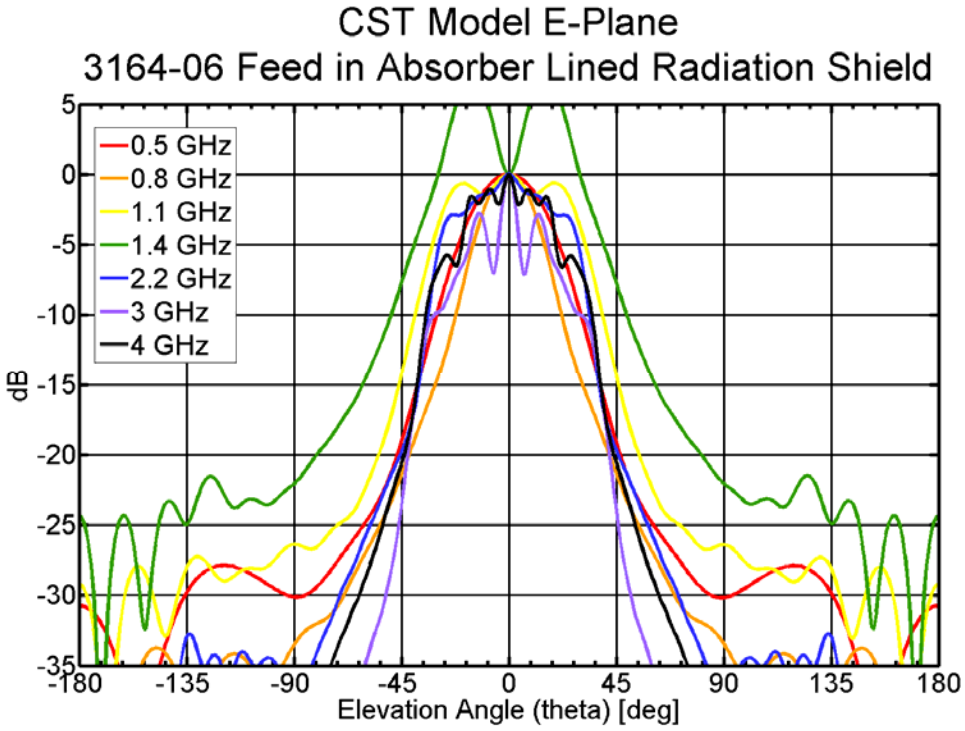
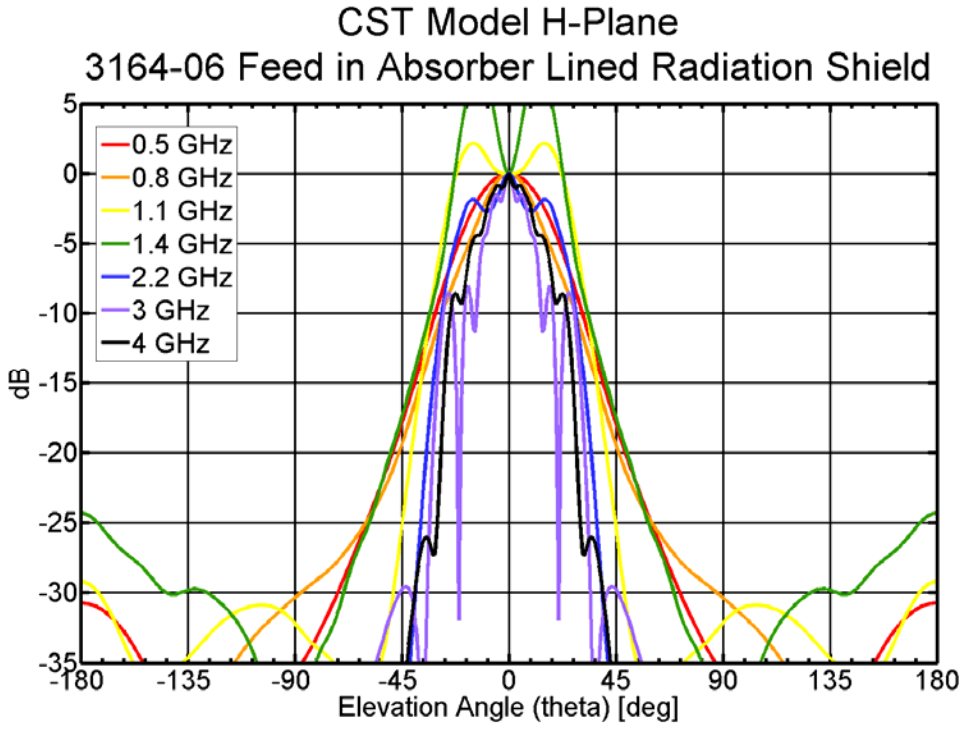


Figure 6.4.4 a

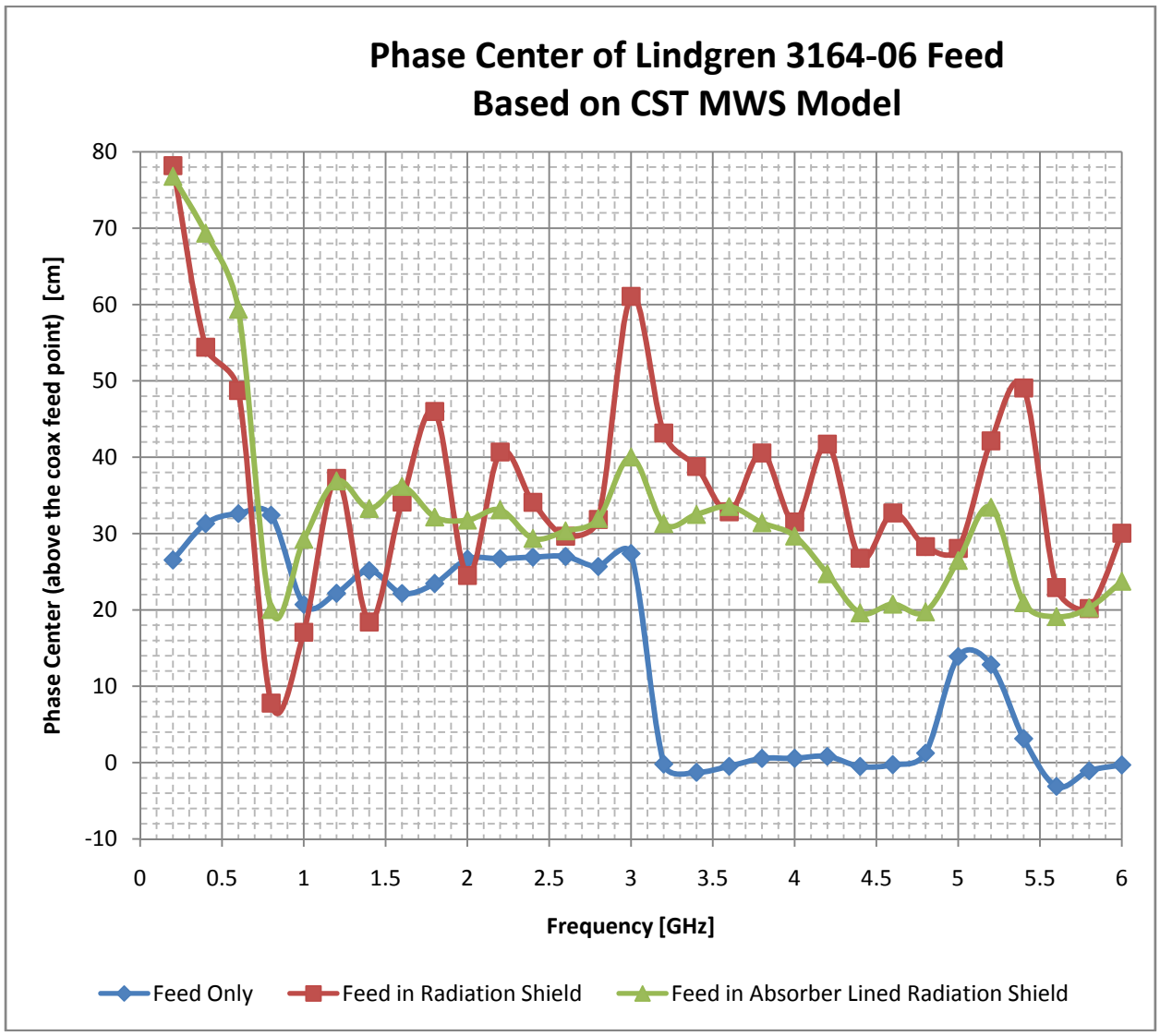
Principal Planes of the CST Model Patterns:



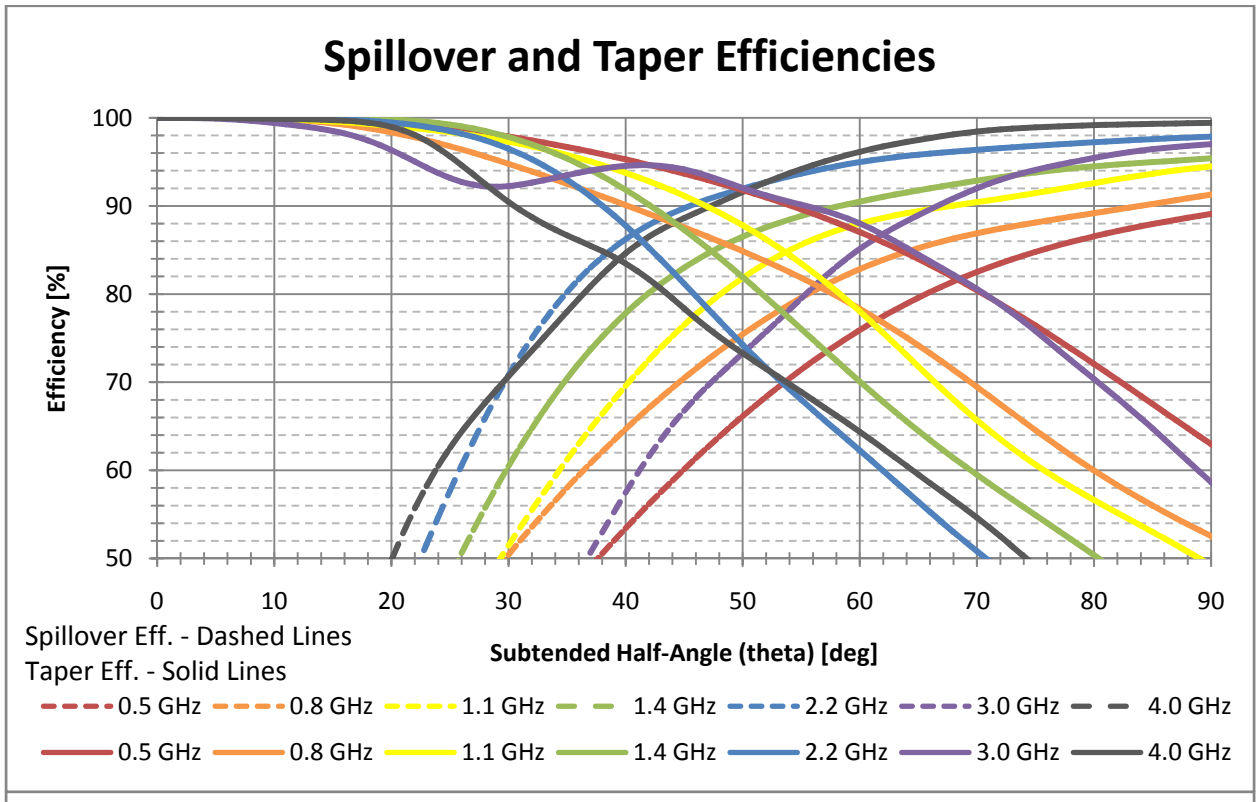


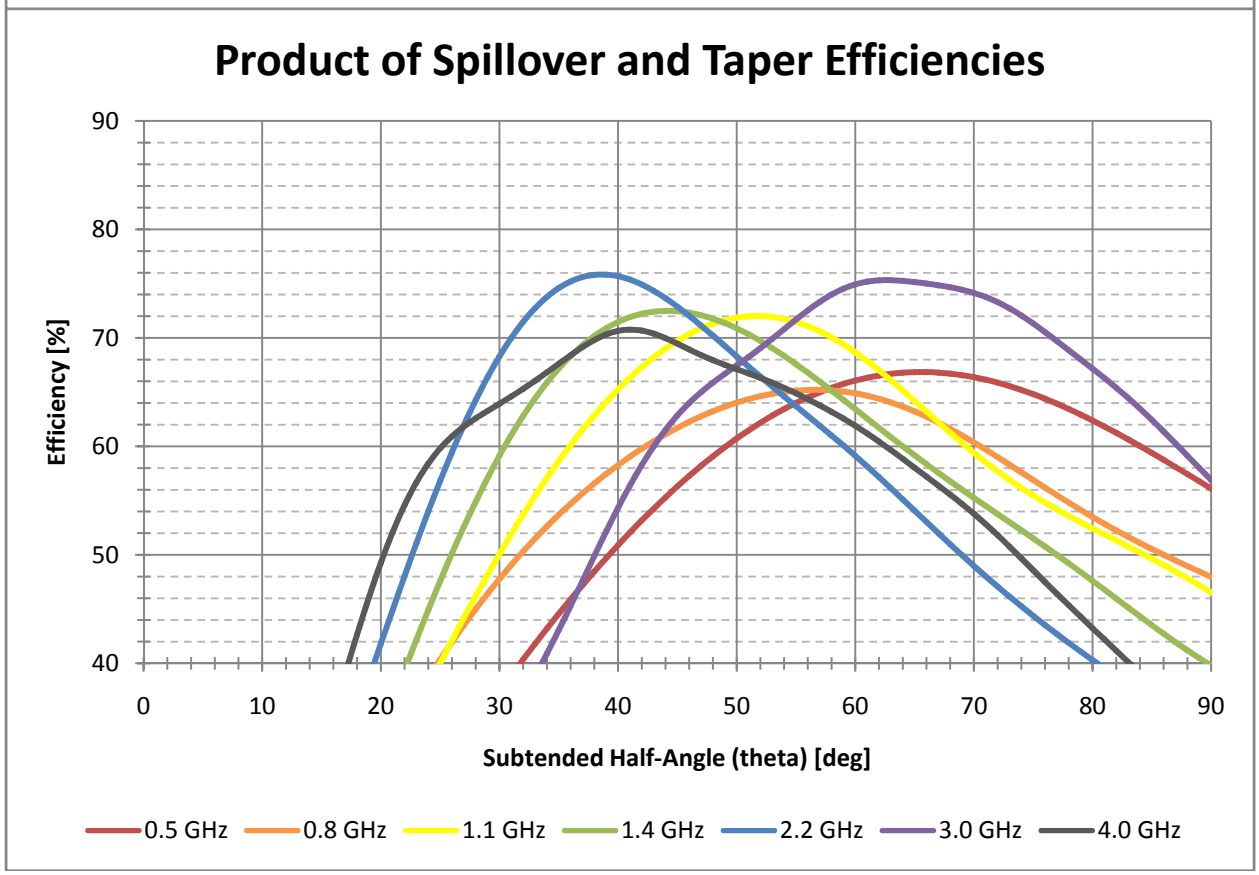
6.4.5 Phase Center

Something may have gone wrong with the Feed Only model. For some unknown reason, the phase center becomes very close to zero at frequencies greater than about 3.0 GHz. The phase center variation appears to be best using the absorber lined radiation shield, where the optimum value for the phase center appears to be about 32 cm.

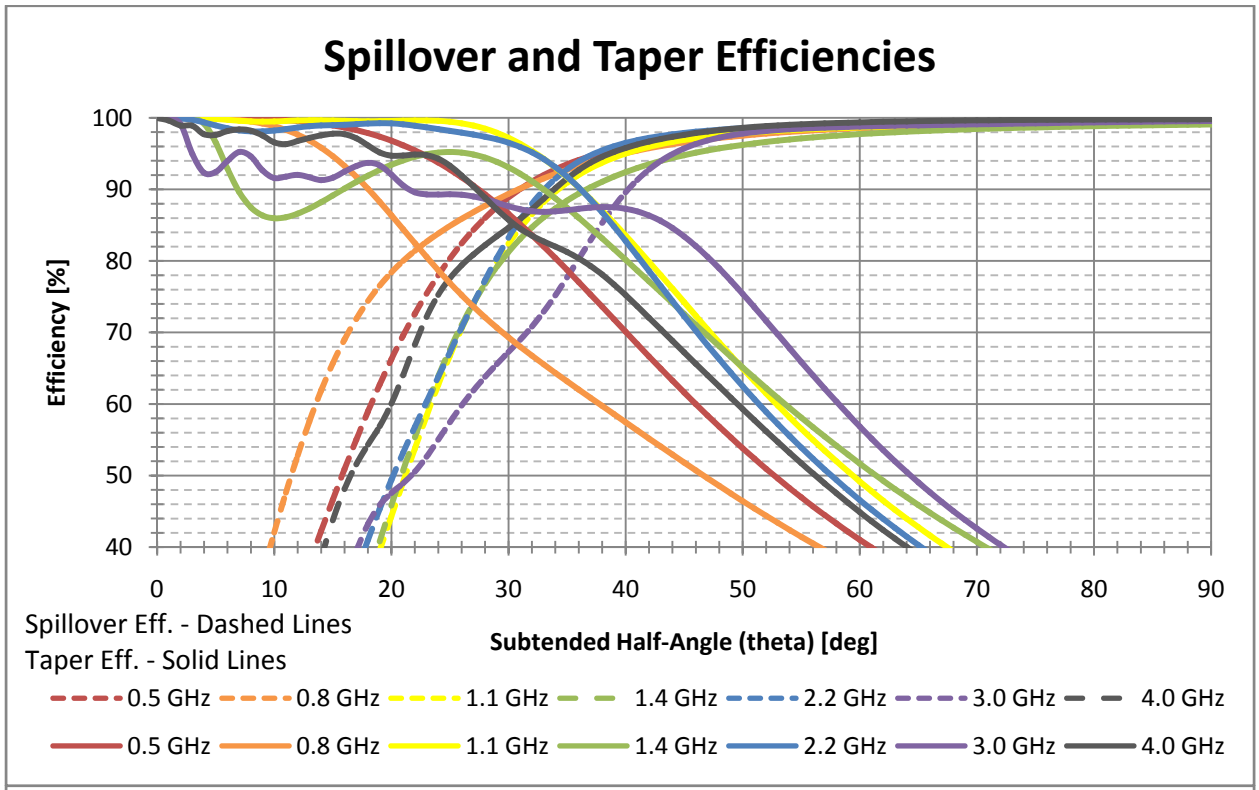


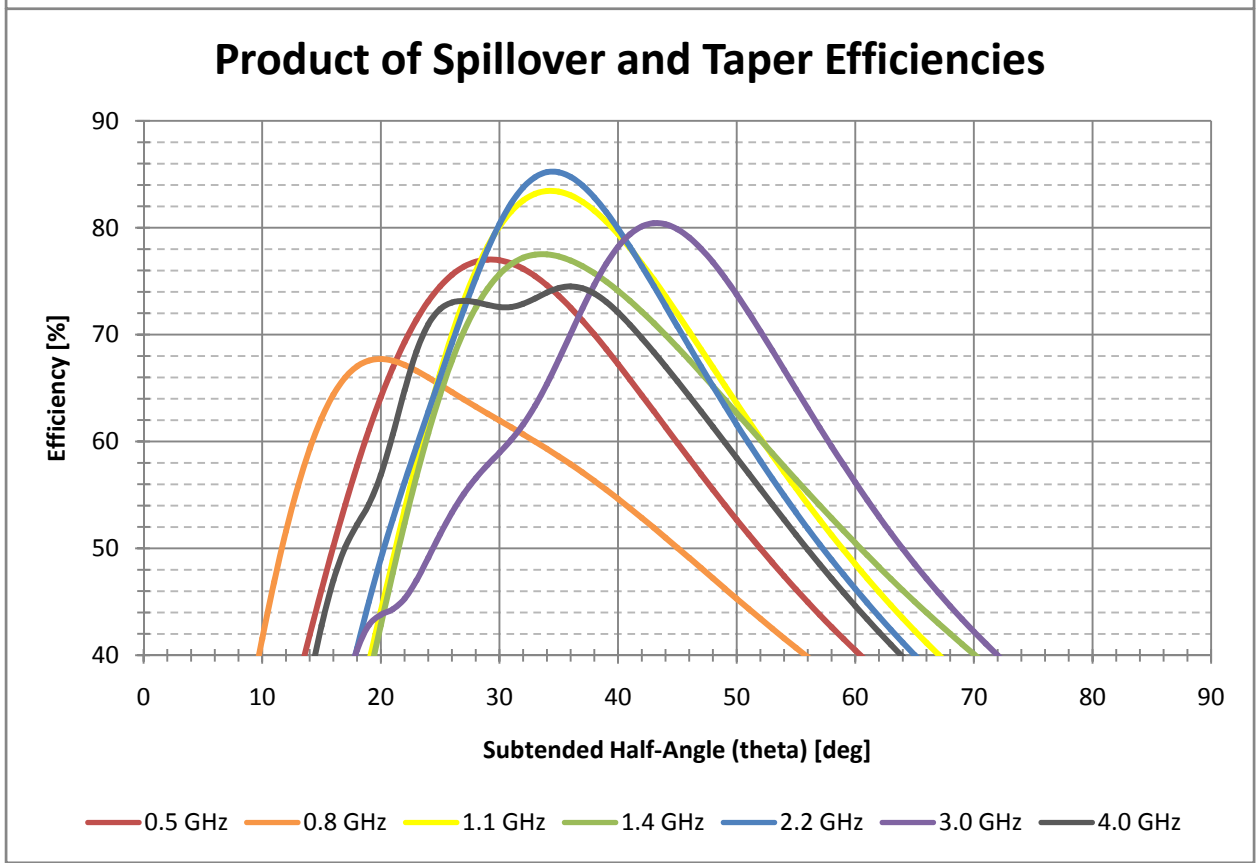
6.4.6 Efficiencies for Feed Only



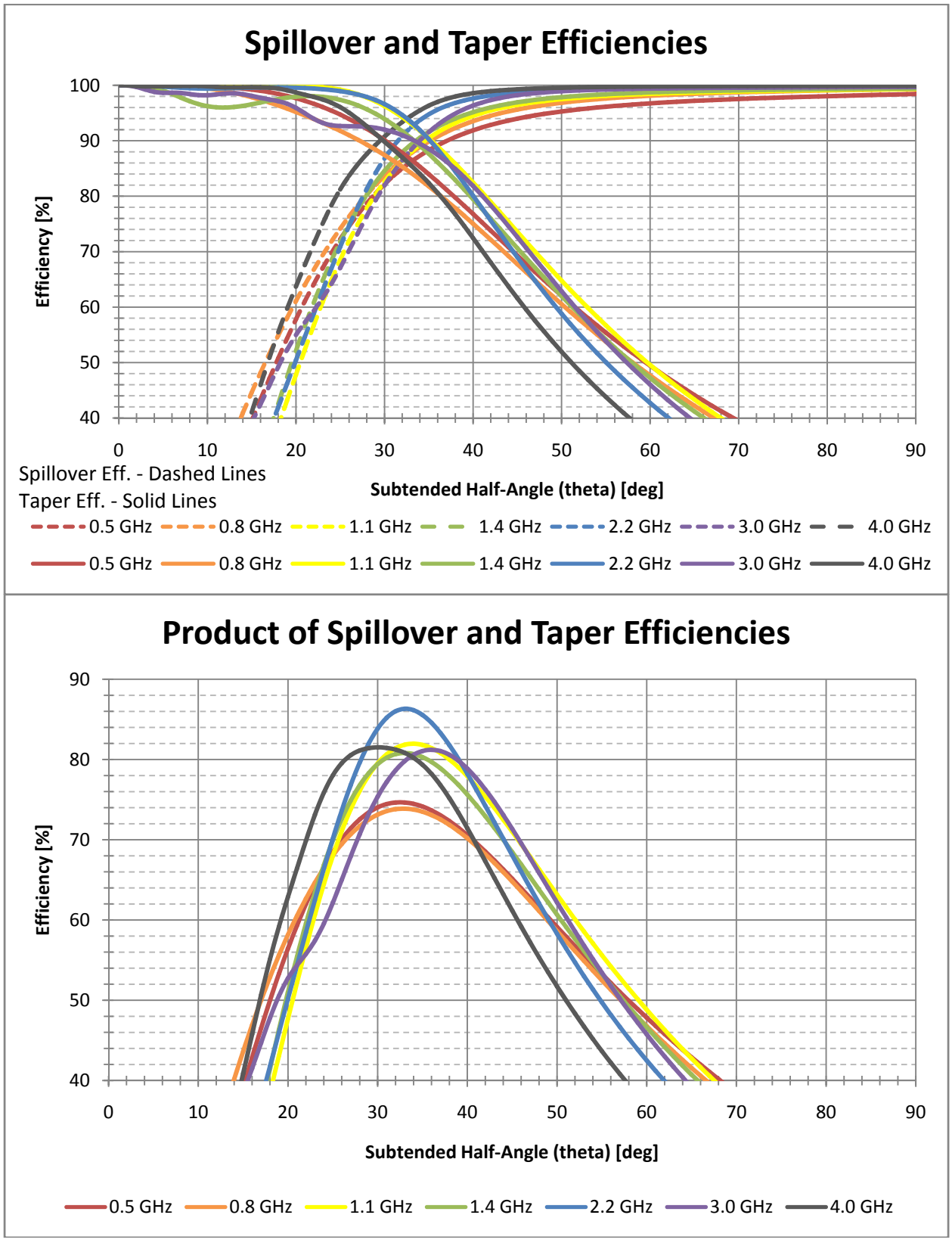


6.4.7 Efficiencies for Feed in Radiation Shield





6.4.8 Efficiencies for Feed in Absorber Lined Radiation Shield



6.5 Effect of Fin Thickness

Several modifications to the Inventor file design of the ETS-Lindgren 3164-05 (see section 3.1.1) were modeled and simulated using QuickWave. These modifications were intended to determine the effect of the fin thickness. Four different structures are discussed in this section:

- 3.615 mm fin (Lindgren Inventor file design)
- 1.372 mm fin
- 1.372 mm fin with 3.615 mm at slot edge
- 1.372 mm fin with 3.615 mm along whole front edge

Each of these designs are described and then the far-field patterns are presented. Finally, comparisons of far-field patterns from each design are shown. Since these structures were designed in QuickWave, accurate return loss data is not available. However, it is expected that the return loss varies greatly between these designs.

6.5.1 Far-Field Patterns for 3.615 mm Fin

This is the design found in the Lindgren Inventor files (see section 3.1.1). The full fin is 3.615 mm wide, and the edge has a straight 45° bevel inside the slot to prevent the fins from touching each other. See the following drawings from QuickWave of the whole structure and fin edge shape.

3.615 mm Fin Design

Figure 6.5.1 a

Top View of Fin Edge at Slot

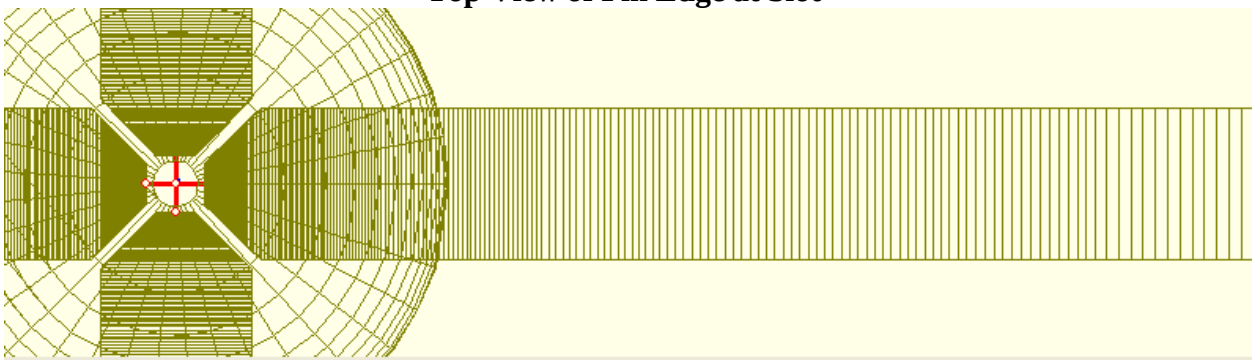
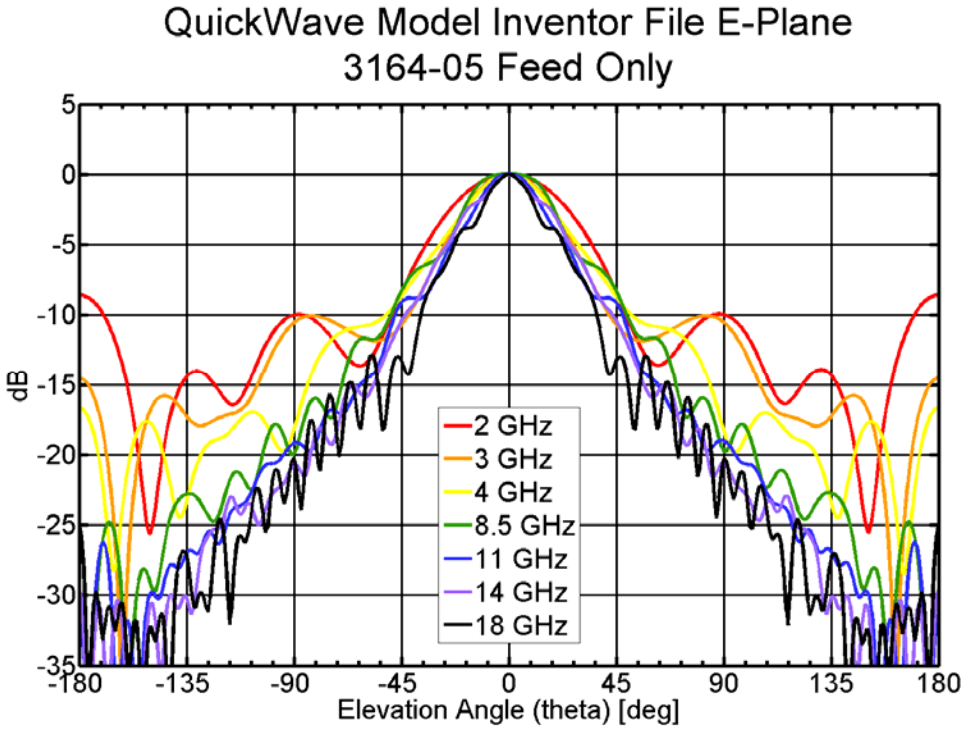
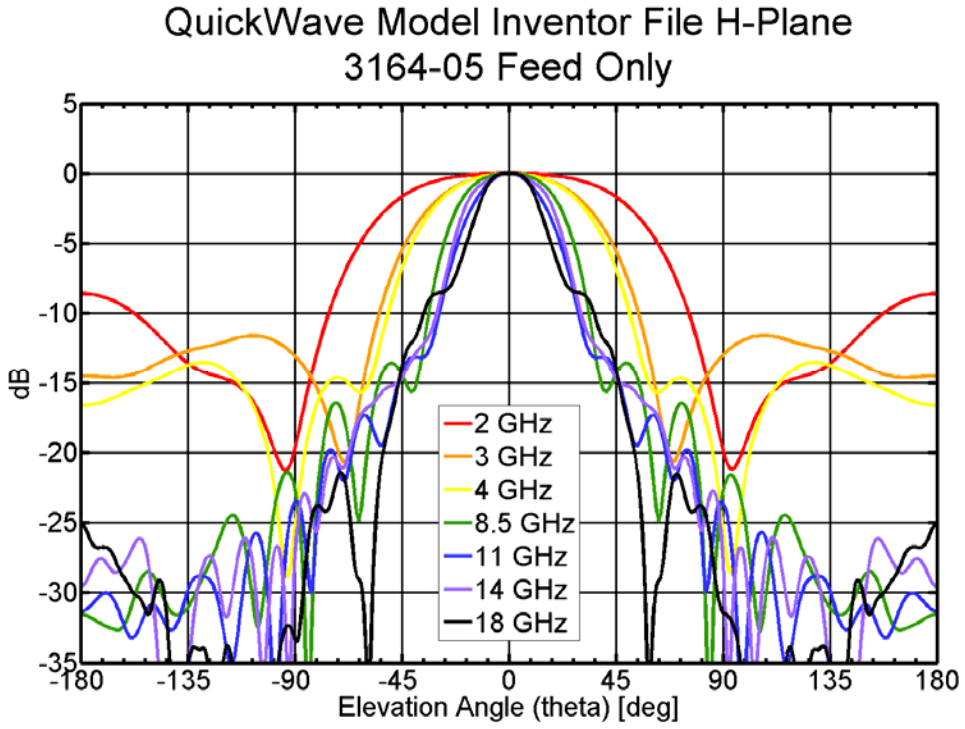


Figure 6.5.1 b

Principal Planes of the 3.615 mm Fin Far-field Patterns:





6.5.2 Far-Field Patterns for 1.372 mm Fin

In this design, the full fin is 1.372 mm wide, and the original 45° edge bevel is retained. The impedance match is changed do to the change in the slot. See the following drawings from QuickWave of the whole structure and fin edge shape.

1.372 mm Fin Design

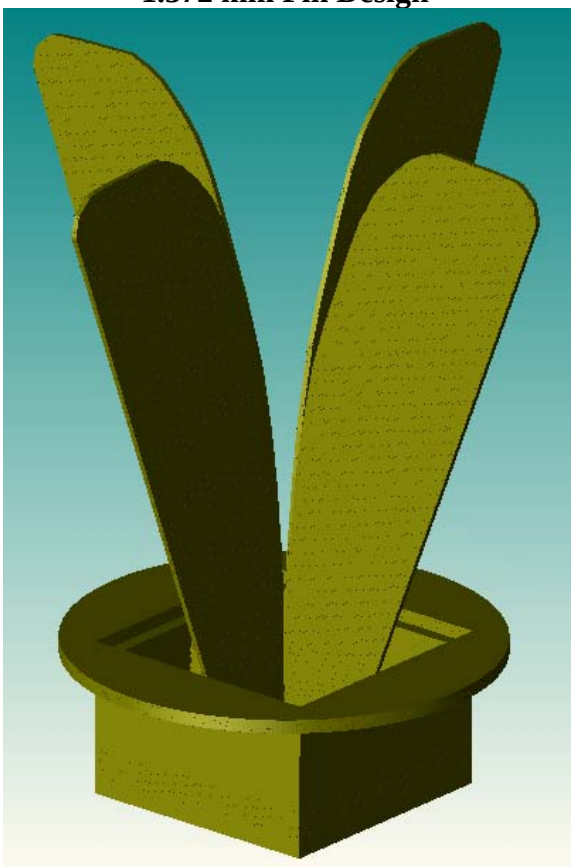


Figure 6.5.2 a

Top View of Fin Edge at Slot

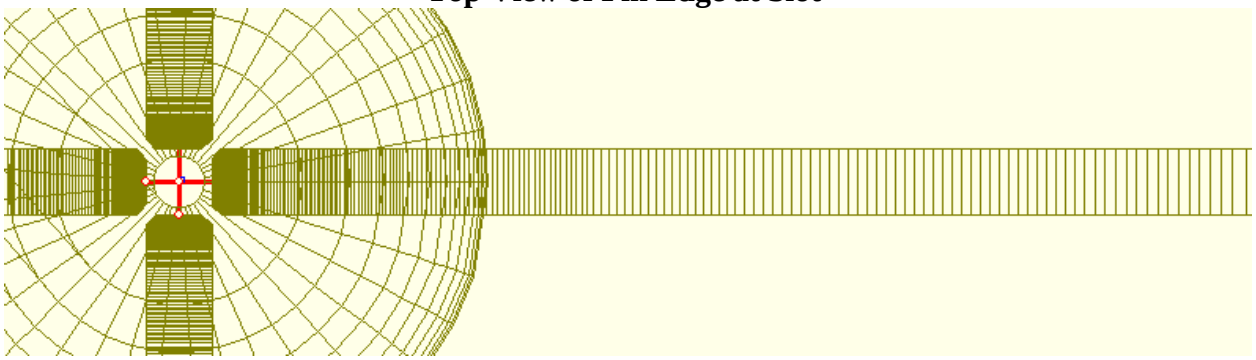
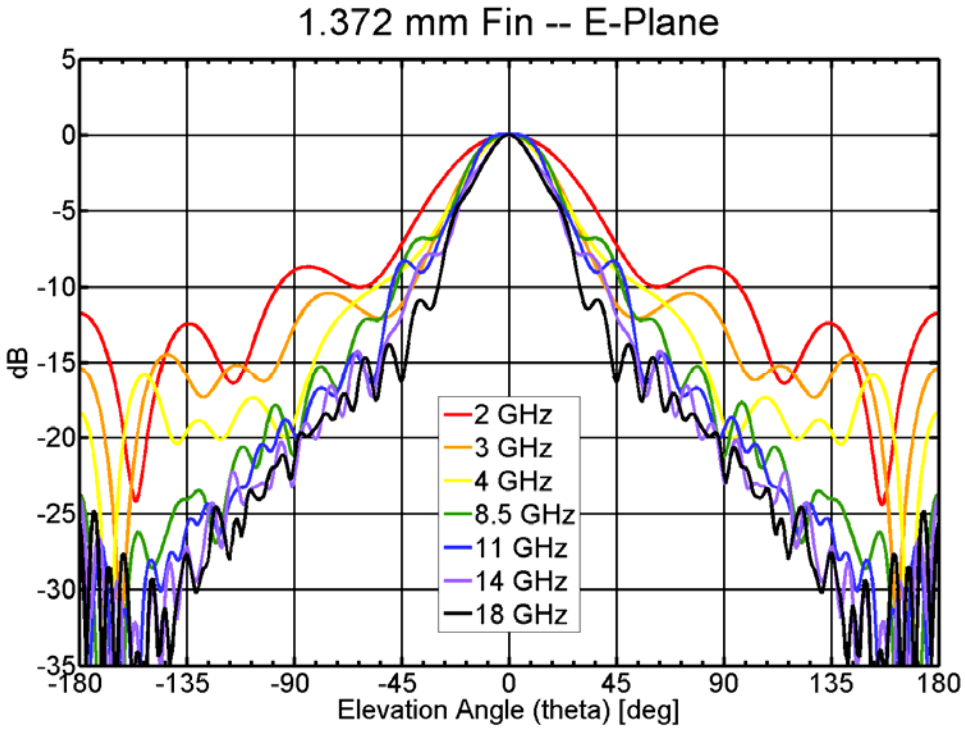
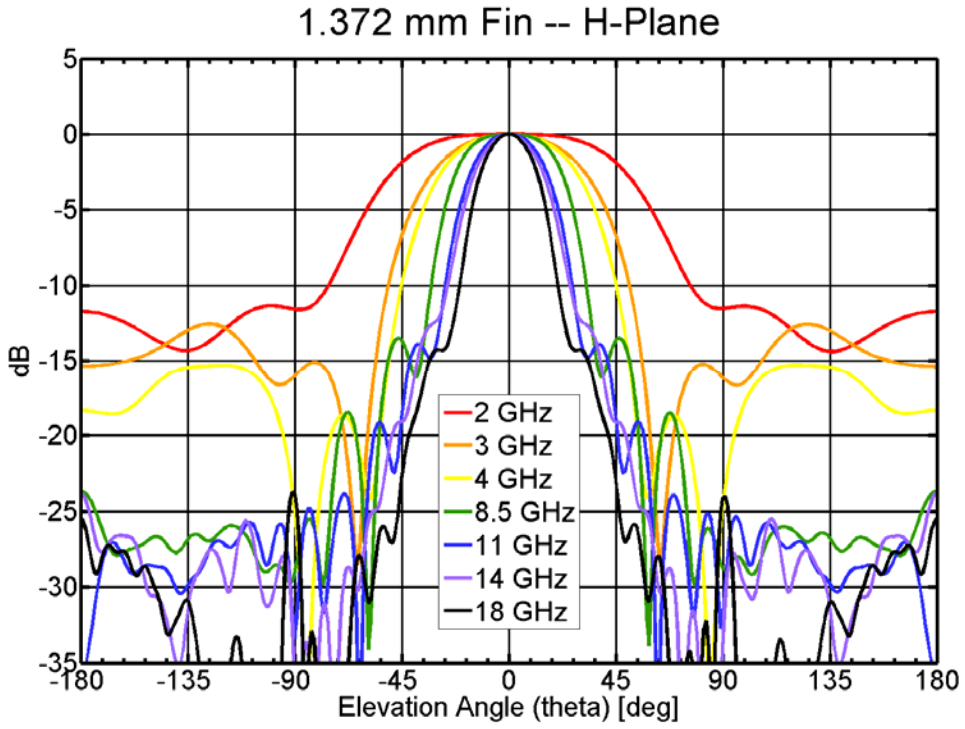


Figure 6.5.2 b

Principal Planes of the 1.372 mm Fin Far-field Patterns:





6.5.3 Far-Field Patterns for 1.372 mm Fin with 3.615 mm at Slot

In this design, the fin is 3.615 mm thick from the slot to a vertical plane 0.770 mm from the edge of the bevel, and the remainder (which is the majority) of the fin is 1.372 mm thick. The 45° edge bevel is retained. The impedance match should be very similar to the Lindgren design, because fin edge is the same in the slot. See the following drawings from QuickWave of the whole structure and fin edge shape. The red indicates the thick part of the fin.

1.372 mm Fin with 3.615 mm at Slot Edge Design

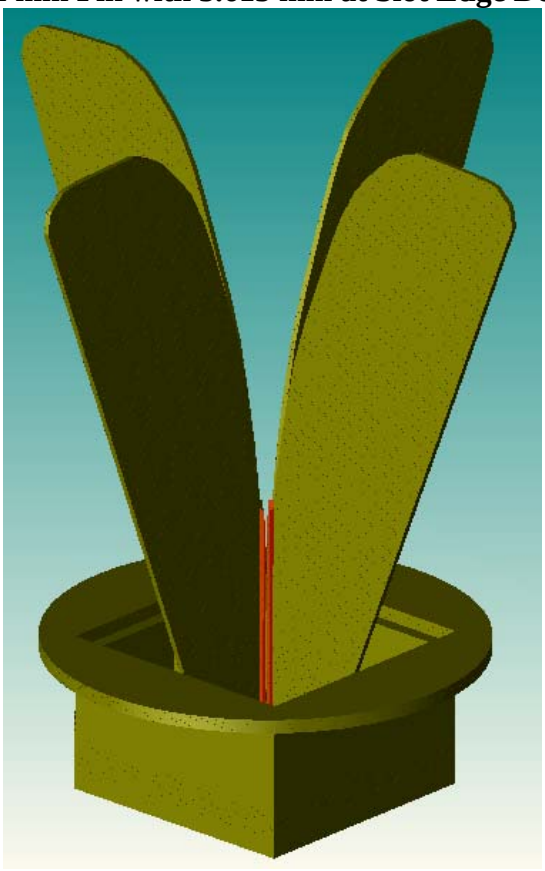


Figure 6.5.3 a

Top View of Fin Edge at Slot

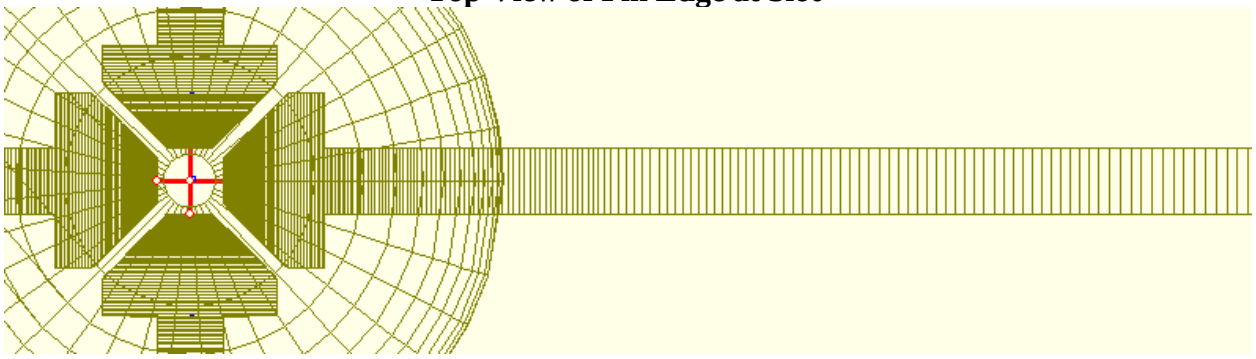
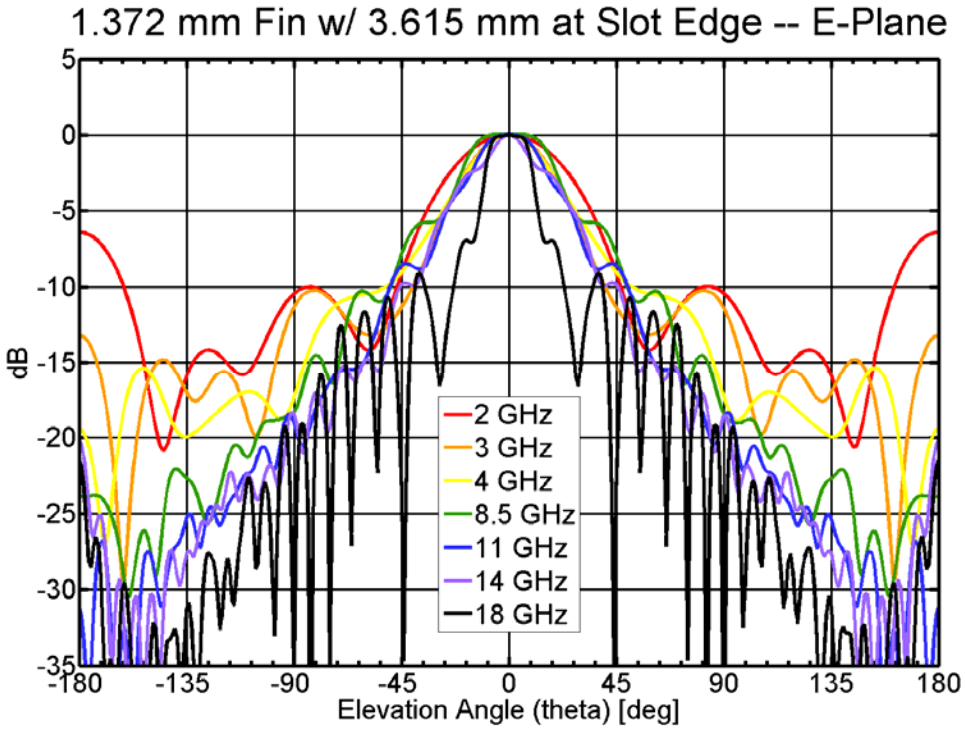
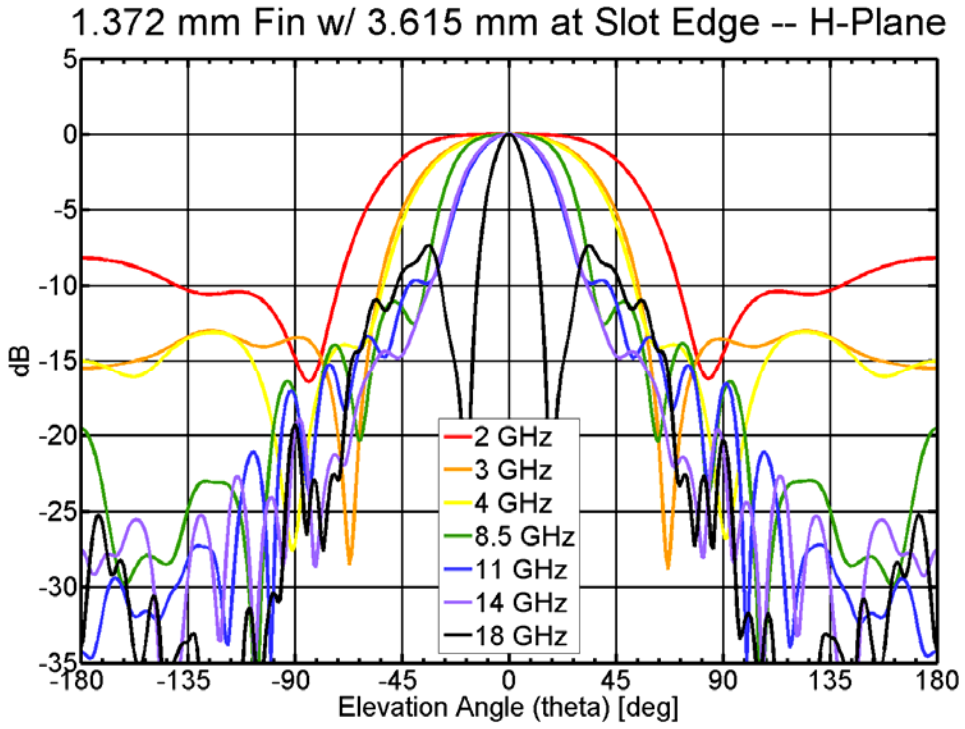


Figure 6.5.3 b

Principal Planes of the 1.372 mm Fin with 3.615 mm at Slot Edge Far-field Patterns:





6.5.4 Far-Field Patterns for 1.372 mm Fin with 3.615 mm along Edge

In this design, the fin is 3.615 mm thick for 1 mm along the entire length of the front edge (including the beveled slot). The remainder (which is the majority) of the fin is 1.372 mm thick. The impedance match and radiation patterns should be very similar to the Lindgren design, because fin edge is the same. See the following drawings from QuickWave of the whole structure and fin edge shape. The red indicates the thick part of the fin. The blue indicated the main bulk of the fin.

1.372 mm Fin with 3.615 mm along Edge Design

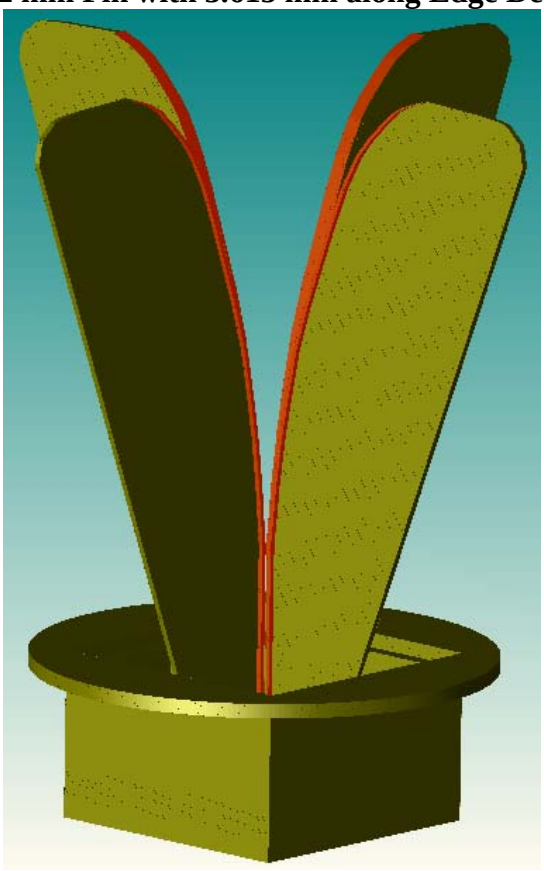


Figure 6.5.4 a

Top View of Fin Edge at Slot

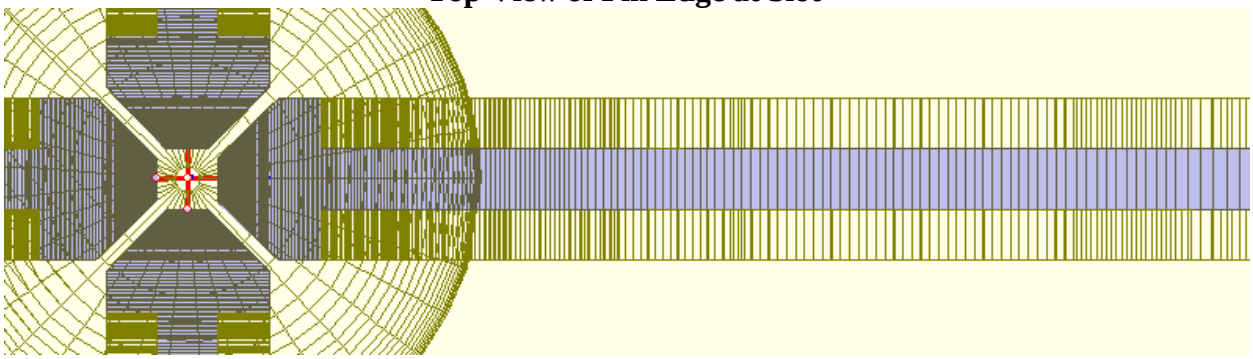
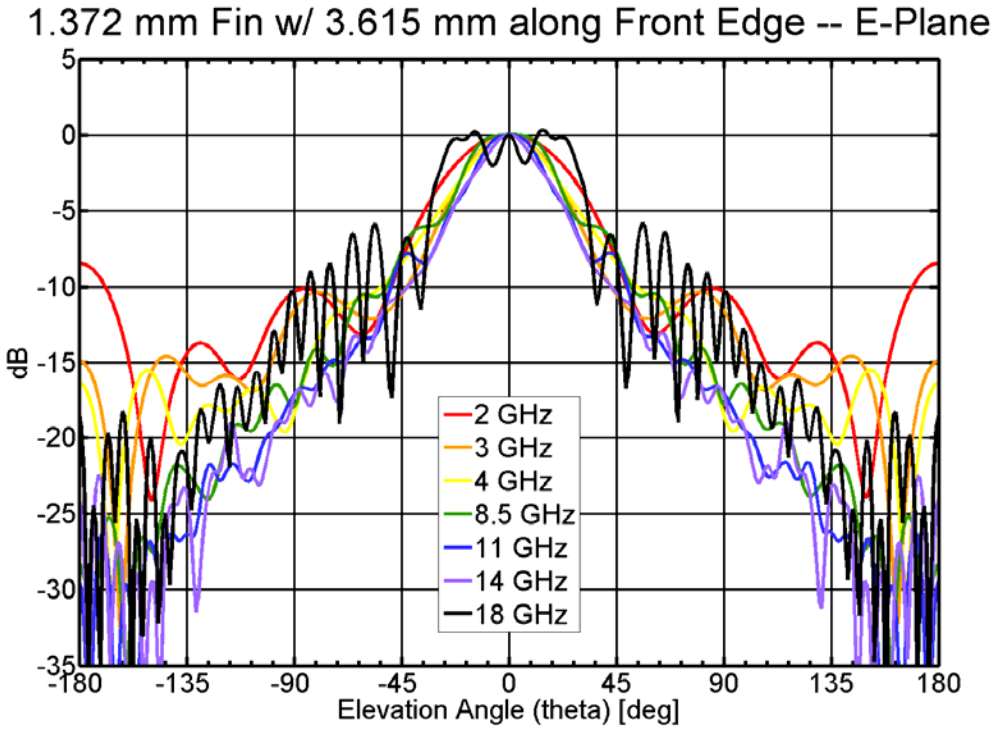
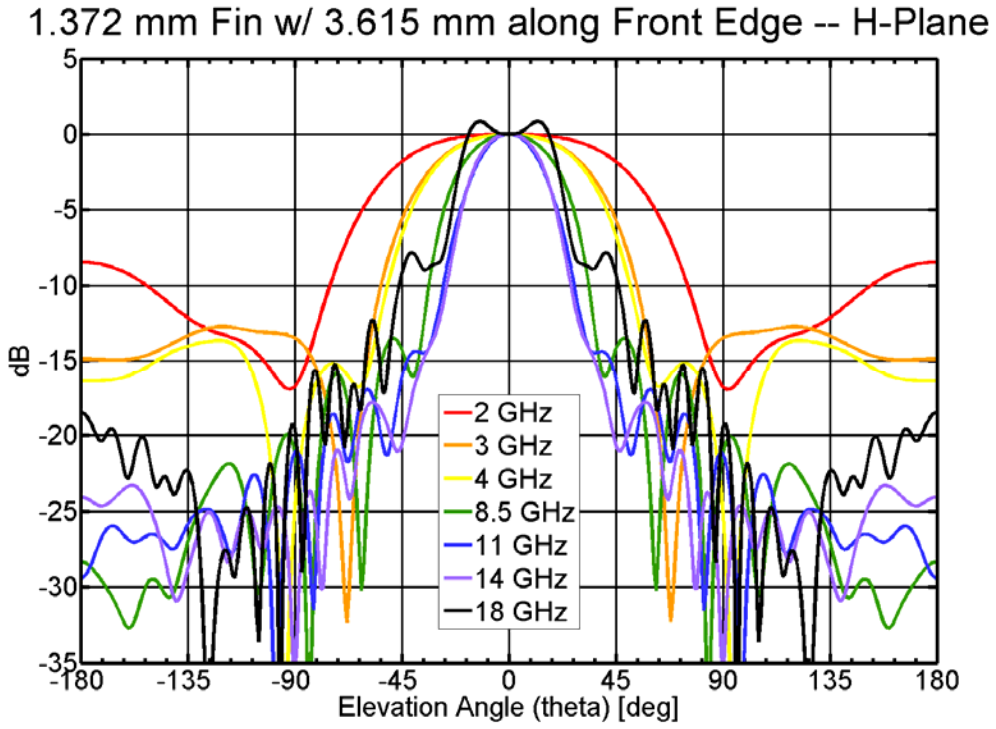


Figure 6.5.4 b

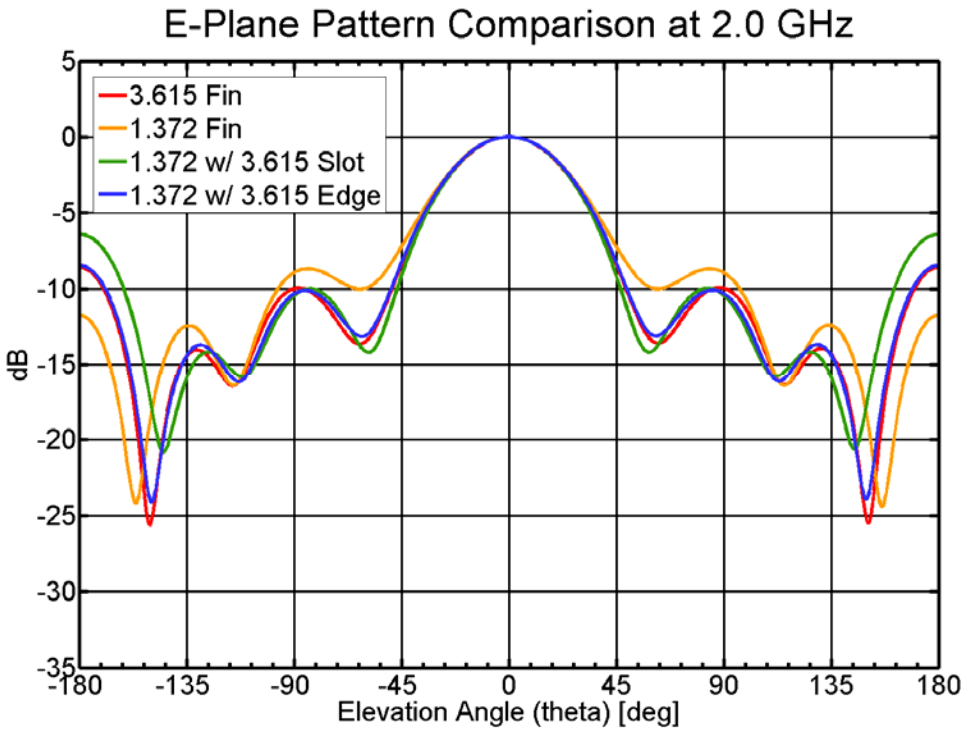
Principal Planes of the 1.372 mm Fin with 3.615 mm along Edge Far-field Patterns:

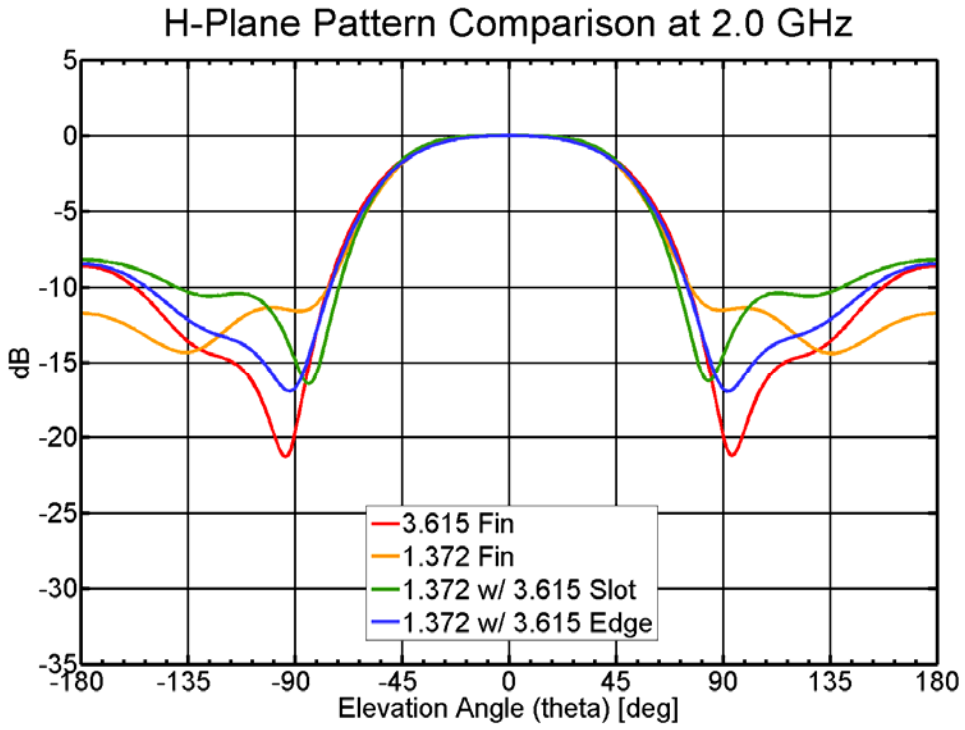




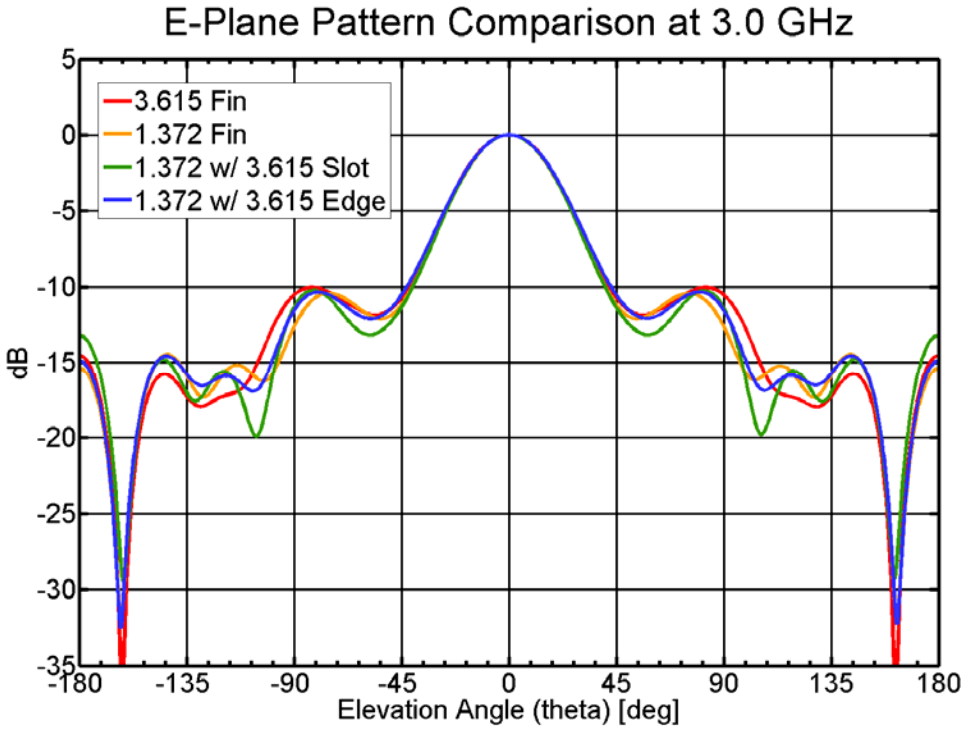
6.5.5 Comparison of Far-field Patterns

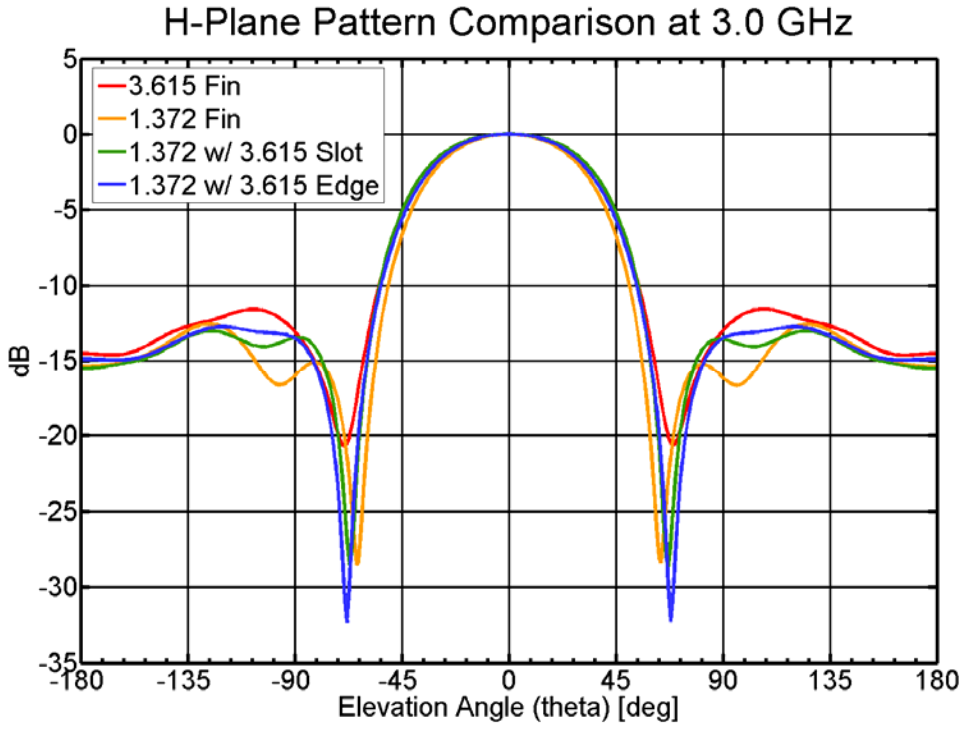
Far-field comparison of principal planes at 2.0 GHz:



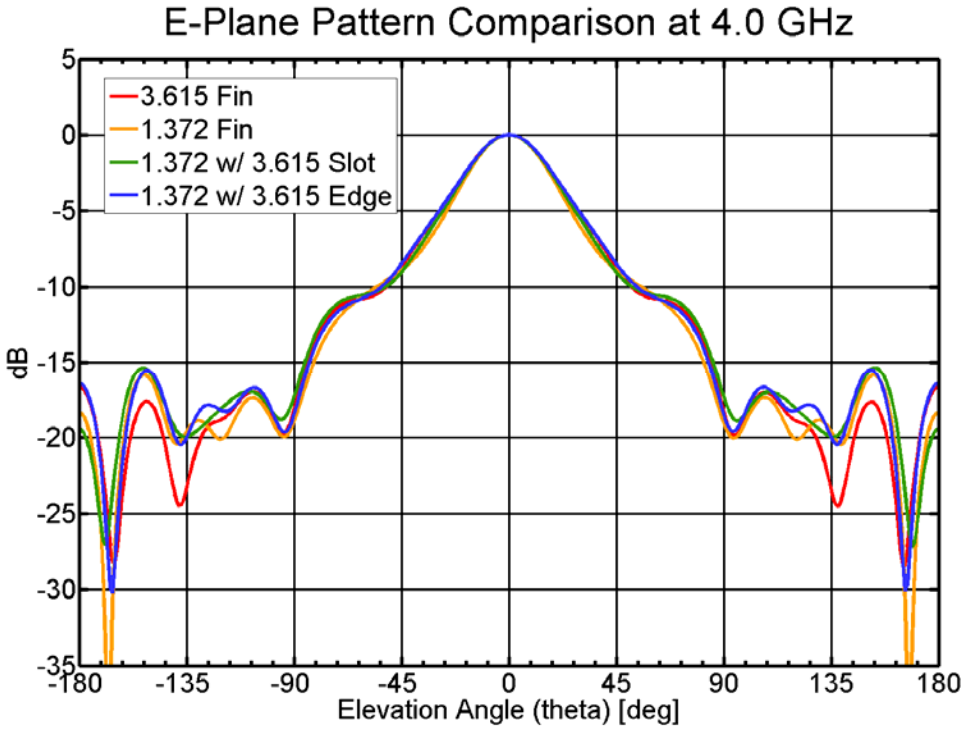


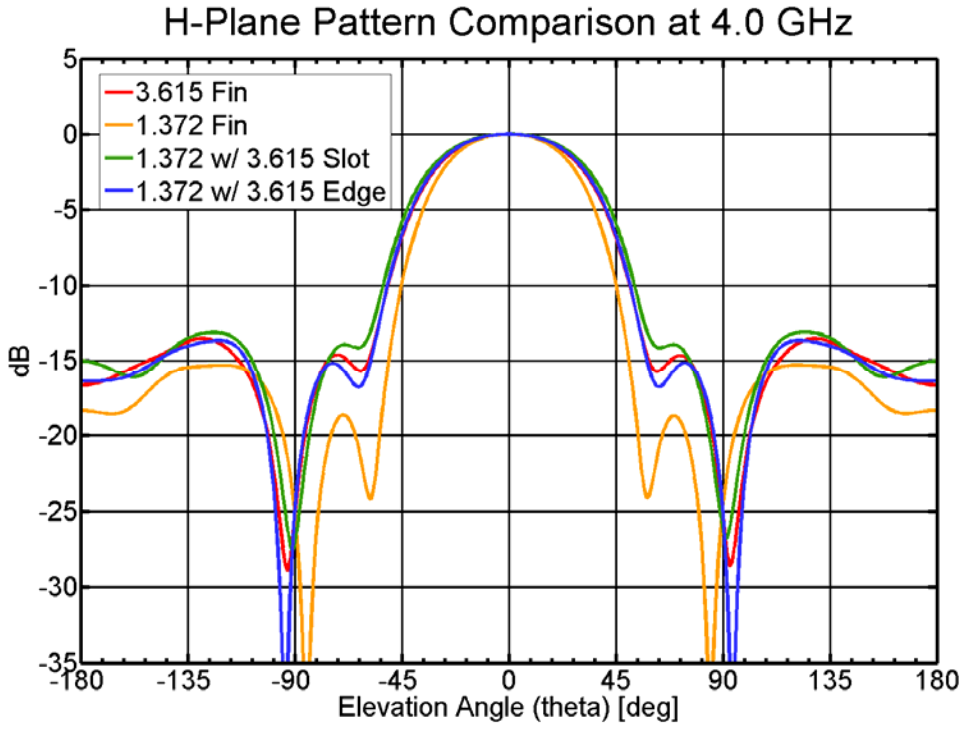
Far-field comparison of principal planes at 3.0 GHz:



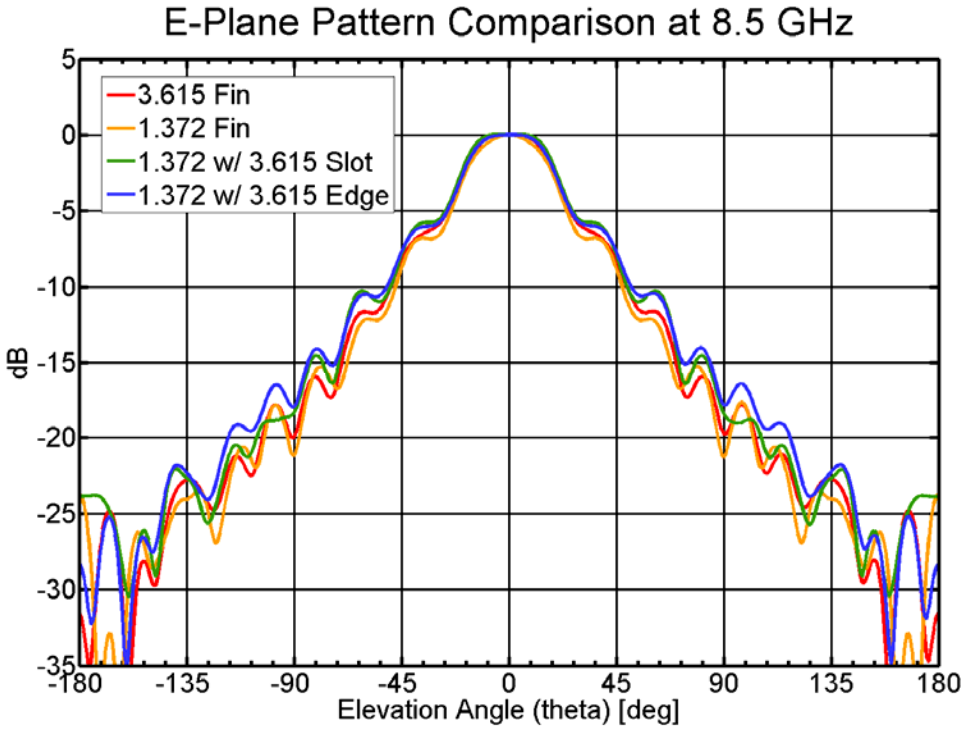


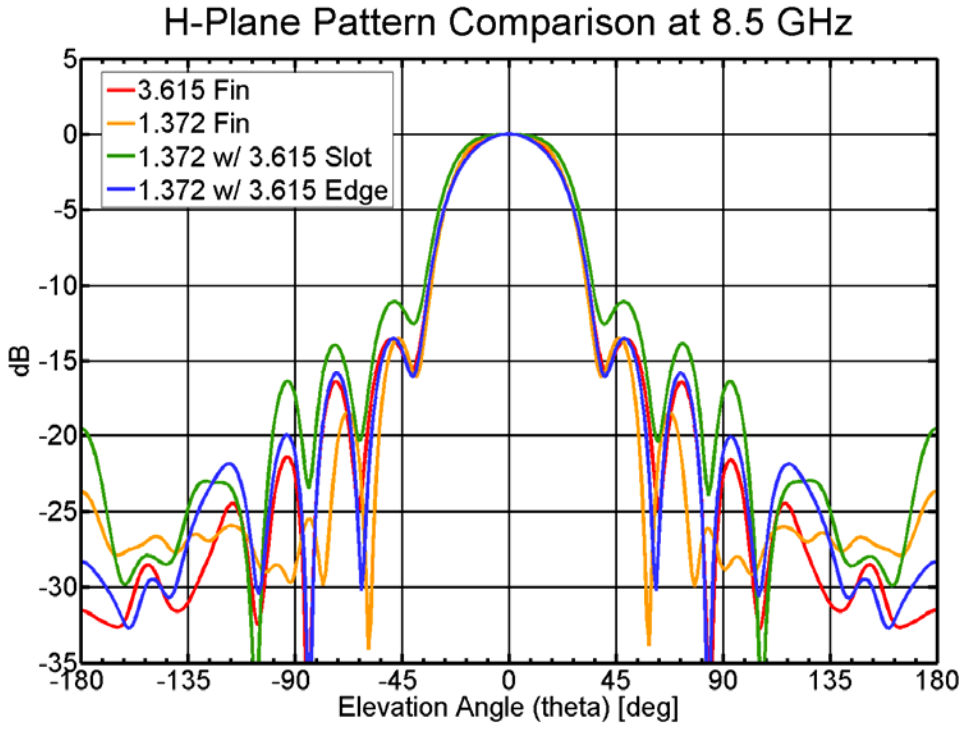
Far-field comparison of principal planes at 4.0 GHz:



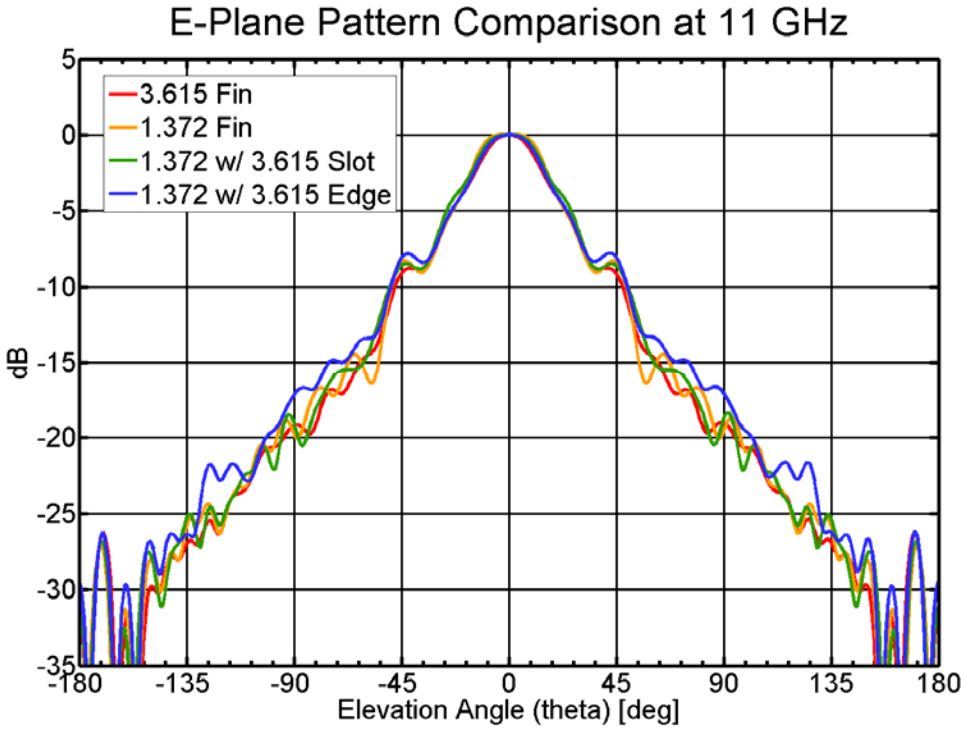


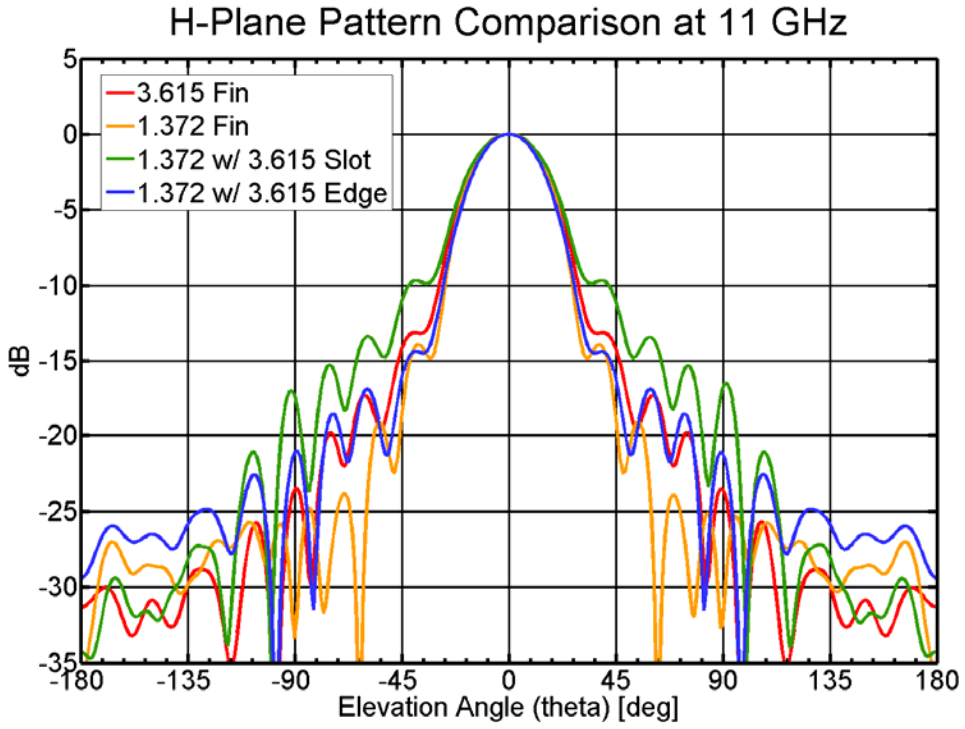
Far-field comparison of principal planes at 8.5 GHz:



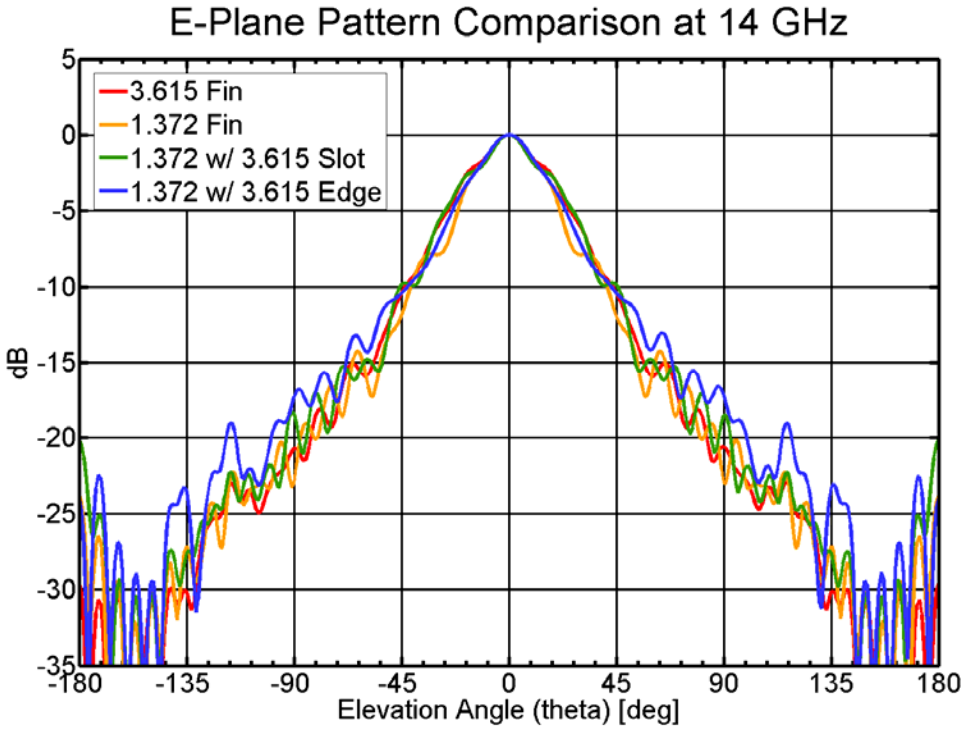


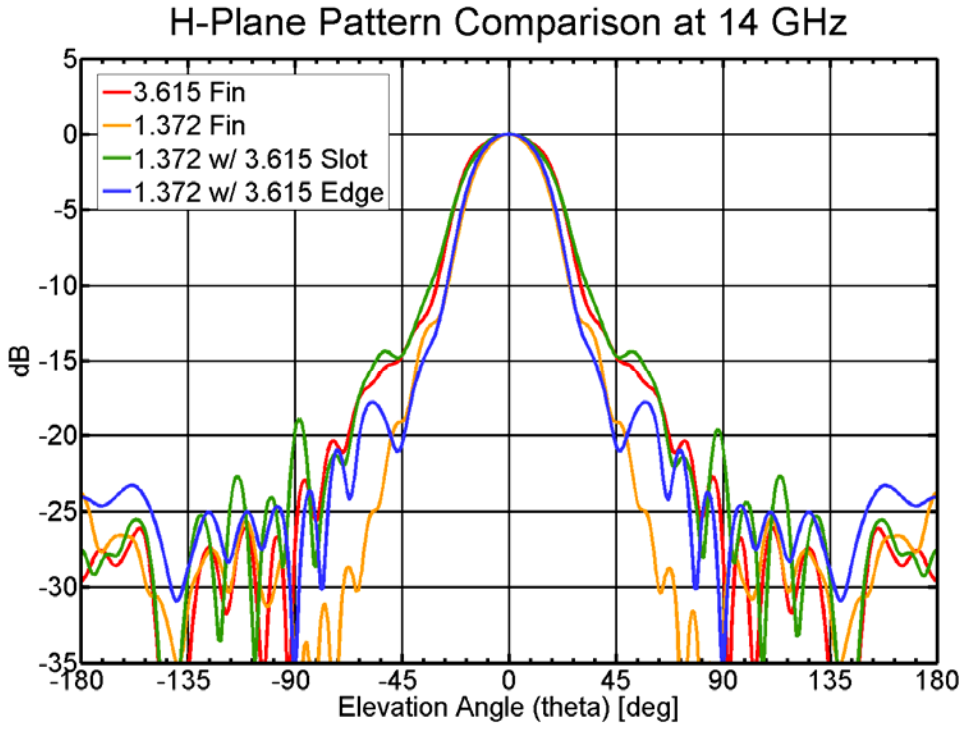
Far-field comparison of principal planes at 11 GHz:



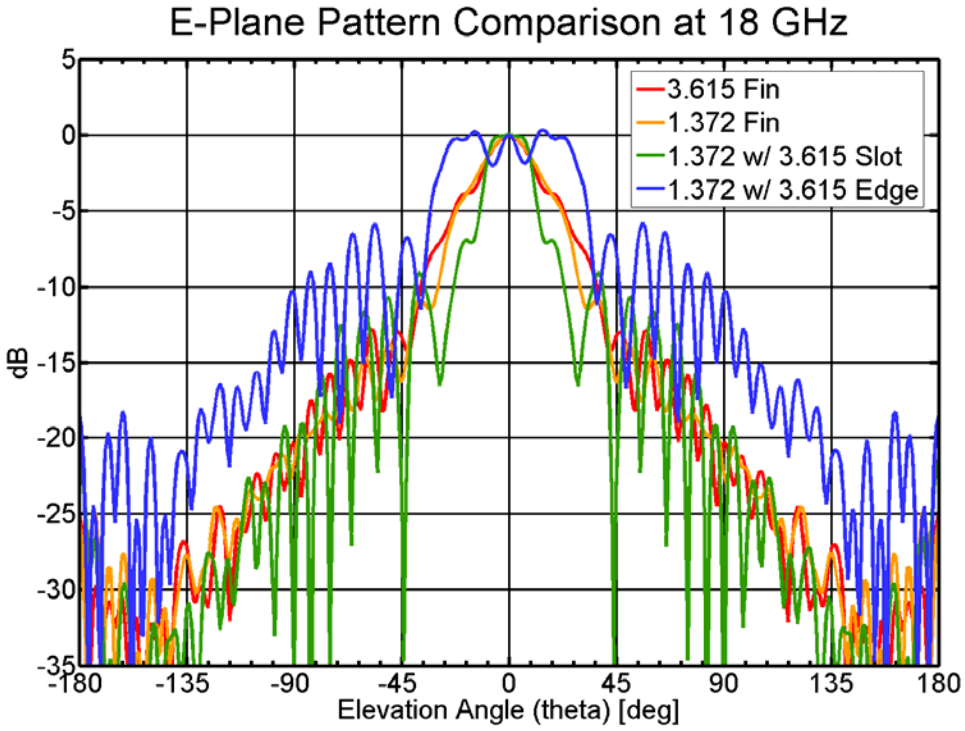


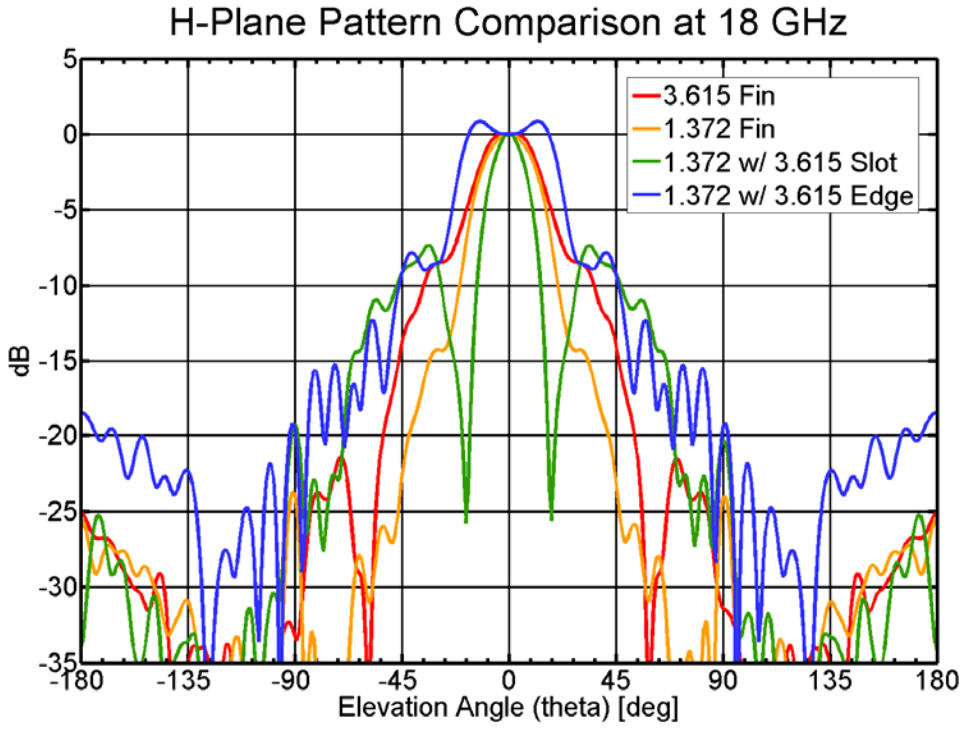
Far-field comparison of principal planes at 14 GHz:





Far-field comparison of principal planes at 18 GHz:





7

References

1 Background on Lindgren Quad-Ridge Horn Feeds

[1-1] Rodriguez, V. (2006 March). A multi-octave open-boundary quadridge horn antenna for use in the S to Ku-bands. *Microwave Journal.* 84-92.

[1-2] Lewis Center for Educational Research. (2007). *Goldstone Apple Valley Radio Telescope (GAVRT).* Retrieved Sept. 18, 2007, from http://www.lewiscenter.org/gavrt/.

[1-3] Imbriale, W. A., Weinreb, S., & Mani, H. (2007 March). Design of Wideband Radio Telescope. *IEEE 2007 Aerospace Conference.* Big Sky, Montana.

[1-4] ETS-Lindgren. (2005 June). *Open-Boundary Quad-Ridge Horns: Model 3164-05 Data Sheet.*

[1-5] ETS-Lindgren. (2007 March). *Model 3164 Series: Open-Boundary Quad-Ridge Horns: User Guide.*

[1-6] ETS-Lindgren. (2007 March). *Open-Boundary Quad-Ridge Horns: Model 3164-06 Data Sheet.*

2 Modeling Software

[2-1] Rudnicki, J. (2007, June 25). *QWED – Software for Electromagnetic Design.* Retrieved Sept 10, 2007, from http://www.qwed.com.pl.

[2-2] QWED. (2007). *QW-Editor Reference Guide, version 6.5.* Warsaw.

[2-3] QWED. (2006). *QW-Simulator Reference Guide, version 6.5.* Warsaw.

[2-4] QWED. (2007). *QW-User Guide, version 6.5.* Warsaw.

[2-5] Computer Simulation Technology. (2007). *CST MICROWAVE STUDIO.* Retrieved Sept 11, 2007, from http://www.cst.com/Content/Products/MWS/Overview.aspx.

[2-6] Computer Simulation Technology. (2005, Sept. 14). *CST MICROWAVE STUDIO 2006 – Getting Started.*

[2-7] Computer Simulation Technology. (2005, Oct. 4). *CST MICROWAVE STUDIO 2006 – Tutorials.*

3 CAD Models of Lindgren Feeds

[3-1] Emerson & Cuming Microwave Products (2007). *ECCOSORB AN.* Retrieved Sept 11, 2007, from http://www.eccosorb.com/products/2/ECCOSORB%20AN.

4 Simulation Outputs

[4-1] QWED. (2007). *QW-Editor Reference Guide, version 6.5.* Warsaw.

[4-2] QWED. (2006). *QW-Simulator Reference Guide, version 6.5.* Warsaw.

[4-3] QWED. (2007). *QW-User Guide, version 6.5.* Warsaw.

[4-4] Space Telescope Science Institute. *Numarray Home Page.* Retrieved Sept 19, 2007, from http://www.stsci.edu/resources/software_hardware/numarray/numarray.html.

[4-5] Computer Simulation Technology. (2005, Sept. 14). *CST MICROWAVE STUDIO 2006 – Getting Started.*

5 External Postprocessing

[5-1] Rudge, A. W., Milne, K., Olver, A. D., & Knight P. (1982). *The Handbook of Antenna Design, Volume 1.* London: Peter Peregrinus on behalf of the Institution of Electrical Engineers.

[5-2] Balanis, C. (1997). *Antenna Theory: Analysis and Design (2nd ed.).* John Wiley & Sons, Inc.